

TEMA 7

INDICE

1. Introducción.....	4
2. Librerías de Java para desarrollar GUIs.....	5
2.1 AWT.....	6
Principales componentes AWT.....	6
2.2 Swing.....	7
Componentes de Swing.....	7
3. Creación de interfaces gráficas de usuario utilizando asistentes y herramientas del entorno integrado.....	14
3.1 Diseñando con asistentes.....	14
4. Eventos.....	19
4.1 Introducción.....	19
¿Qué es un evento?.....	19
¿Qué es la programación guiada por eventos?.....	19
4.2 Modelo de gestión de eventos.....	19
Ejemplo del modelo de gestión de eventos.....	21
4.3 Tipos de eventos.....	24
4.4 Eventos de teclado.....	24
Código del oyente de teclado.....	25
4.5 Eventos de ratón.....	26
4.6 Creación de controladores de eventos.....	29
5. Generación de programas en entorno gráfico.....	30
5.1 Contenedores.....	30
5.2 Cerrar la aplicación.....	31
5.3 Organizadores de contenedores: layout managers.....	31
5.4 Contenedor ligero: JPanel.....	33
5.5 Etiquetas y campos de texto.....	33
Propiedades asociadas a JTextField, y los métodos que permiten modificarlas.....	34
5.6 Botones.....	35
5.7 Casillas de verificación y botones de radio.....	36
5.8 Listas.....	36
5.8.1 Listas (II).....	39
Ejemplo “Copiar una lista a otra”.....	39
5.9 Listas desplegables.....	40
Ejemplo “PruebaJComboBox”.....	41
5.10 Menús.....	42
Ejemplo “PruebaMenu”.....	42
5.10.1 Separadores.....	45
5.10.2 Aceleradores de teclado y mnemónicos.....	46

Comunicándonos con el usuario. Interfaces

CASO.

Ana está cursando el módulo de Programación. En el aula, suele sentarse junto a un compañero: **José Javier**. En clase llevan unos días explicándoles cómo construir aplicaciones Java, utilizando interfaces gráficas de usuario (GUI). Pero, ¿qué es una interfaz de usuario? A grandes rasgos, les han comentado que una interfaz de usuario es el medio con que el usuario puede comunicarse con una computadora. Las interfaces gráficas de usuario incluyen elementos tales como: menús, ventanas, paneles, etc, en general, elementos gráficos que facilitan la comunicación entre el ser humano y la computadora.

Hasta ahora, las aplicaciones de ejemplo que habían realizado, estaban en modo consola, o modo carácter, y están contentos porque están viendo las posibilidades que se les abren ahora. Están comprobando que podrán dar a sus aplicaciones un aspecto mucho más agradable para el usuario.

Ana le comenta a **José Javier**:

-“Así podremos dar un aspecto profesional a nuestros programas”.

1. Introducción.

Hoy en día las **interfaces** de los programas son cada vez más sofisticadas y atractivas para el usuario. Son intuitivas y cada vez más fáciles de usar: pantallas táctiles, etc.

Sin embargo, no siempre ha sido así. No hace muchos años, antes de que surgieran y se popularizaran las interfaces gráficas de usuario para que el usuario interactuara con el sistema operativo con sistemas como Windows, etc, se trabajaba en **modo consola**, o **modo carácter**, es decir, se le daban las ordenes al ordenador con comandos por teclado, de hecho, por entonces no existía el ratón.

Así que, con el tiempo, con la idea de simplificar el uso de los ordenadores para extender el uso a un cada vez mayor espectro de gente, de usuarios de todo tipo, y no sólo para los expertos, se ha convertido en una práctica habitual utilizar **interfaces gráficas de usuario (IGU ó GUI** en inglés), para que el usuario interactúe y establezca un contacto más fácil e intuitivo con el ordenador

En ocasiones, verás otras definiciones de interfaz, como la que define una interfaz como un dispositivo que permite comunicar dos sistemas que no hablan el mismo lenguaje. También se emplea el término interfaz para definir el juego de conexiones y dispositivos que hacen posible la comunicación entre dos sistemas.

Aquí en el módulo, cuando hablamos de interfaz nos referimos a la cara visible de los programas tal y como se presenta a los usuarios para que interactúen con la máquina. La interfaz gráfica implica la presencia de un monitor de ordenador, en el que veremos la interfaz constituida por una serie de **menús e iconos que representan las opciones que el usuario** puede tomar dentro del sistema

Las interfaces gráficas de usuario proporcionan al usuario ventanas, cuadros de diálogo, barras de herramientas, botones, listas desplegables y muchos otros elementos. Las aplicaciones son conducidas por eventos y se desarrollan haciendo uso de las clases que para ello nos ofrece el API de Java.

En 1981 aparecieron los primeros ordenadores personales, los llamados PCs, pero hasta 1993 no se generalizaron las interfaces gráficas de usuario. El escritorio del sistema operativo Windows de Microsoft y su sistema de ventanas sobre la pantalla se ha estandarizado y universalizado, pero fueron los ordenadores Macintosh de la compañía Apple los pioneros en introducir las interfaces gráficas de usuario.

En el siguiente enlace puedes ver la evolución en las interfaces gráficas de diverso software, entre ellos, el de las GUI de MAC OS, o de Windows en las sucesivas versiones
<http://www.guidebookgallery.org/screenshots>

Señala la opción correcta acerca de las interfaces gráficas de usuario:



Son sinónimos de ficheros de texto.



Las ventanas de una aplicación no serían un ejemplo de elemento de interfaz gráfica de usuario.



Surgen con la idea de facilitar la interacción del usuario con la máquina.



Ninguna es correcta.

2. Librerías de Java para desarrollar GUIs.

CASO.

José Javier está un poco abrumado por la cantidad de componentes que tiene Java para desarrollar interfaces gráficas. Piensa que hay tantos, que son tantas clases, que nunca podrá aprendérselos.

Ana le dice:

“José Javier, no te preocupes, seguro que haciendo programas, enseguida ubicarás los que más utilices, y el resto, siempre tienes la documentación de Java para cuando dudes, consultarla”.

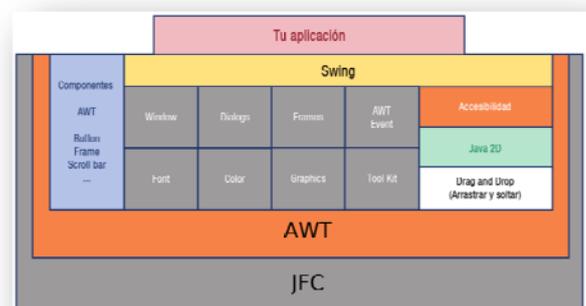
Hemos visto que la interfaz gráfica de usuario es la parte del programa que permite al usuario interactuar con él. Pero, ¿cómo la podemos crear en Java?

El API (Application Programming Interface) de Java proporciona una librería de clases para el desarrollo de interfaces gráficas de usuario (en realidad son dos: **AWT** y **Swing**). Esas librerías se engloban bajo los nombres de **AWT** y **Swing**, que a su vez forman parte de las **Java Foundation Classes** o **JFC**.

Entre las clases de las **JFC** hay un grupo de elementos importante que ayuda a la construcción de interfaces gráficas de usuario (**GUI**) en Java.

Los elementos que componen las JFC son:

- ✓ **Componentes Swing:** encontramos componentes tales como botones, cuadros de texto, ventanas o elementos de menú.
- ✓ **Soporte de diferentes aspectos y comportamientos (Look and Feel):** Permite la elección de diferentes apariencias de entorno. Por ejemplo, el mismo programa puede adquirir un aspecto **Metal** Java (multiplataforma, igual en cualquier entorno), **Motif** (el aspecto estándar para entornos Unix) o **Windows** (para entornos Windows). De esta forma, el aspecto de la aplicación puede ser independiente de la plataforma, lo cual está muy bien para un lenguaje que lleva la seña de identidad de multiplataforma, y se puede elegir el que se desee en cada momento.
- ✓ **Interfaz de programación Java 2D:** permite incorporar gráficos en dos dimensiones, texto e imágenes de alta calidad.
- ✓ **Soporte de arrastrar y soltar (Drag and Drop)** entre aplicaciones Java y aplicaciones nativas (está diseñada para ejecutarse en el entorno informático (lenguaje de máquina y sistema operativo) al que se hace referencia. El término se emplea en contraposición a una aplicación, por ejemplo, escrita en Java que no es nativa, no está diseñada para una sola plataforma). Es decir, se implementa un portapapeles. Llamamos aplicaciones nativas a las que están desarrolladas en un entorno y una plataforma concretos (por ejemplo Windows o Unix), y sólo funcionarán en esa plataforma.
- ✓ **Soporte de impresión.**
- ✓ **Soporte sonido:** Captura, reproducción y procesamiento de datos de audio y MIDI
- ✓ **Soporte de dispositivos de entrada distintos del teclado,** para japonés, chino, etc.
- ✓ **Soporte de funciones de Accesibilidad, para crear interfaces para discapacitados:** Permite el uso de tecnologías como los lectores de pantallas o las pantallas Braille adaptadas a las personas discapacitadas.



Con estas librerías, Java proporciona un conjunto de herramientas para la construcción de interfaces gráficas que tienen una apariencia y se comportan de forma semejante en todas las plataformas en las que se ejecuten.

Señala la opción incorrecta. JFC consta de los siguientes elementos:

- Componentes Swing.
- Soporte de diversos “look and feel”.
- Soporte de impresión.
- Interfaz de programación Java 3D.**

2.1 AWT.

Cuando apareció Java, los componentes gráficos disponibles para el desarrollo de GUI se encontraban en la librería denominada Abstract Window Toolkit (**AWT**).

Las clases AWT se desarrollaron usando código nativo (o sea, código asociado a una plataforma concreta). Eso **dificultaba la portabilidad** de las aplicaciones. Al usar código nativo, para poder conservar la portabilidad era necesario restringir la funcionalidad a los mínimos comunes a todas las plataformas donde se pretendía usar AWT. Como consecuencia, AWT es una librería con una funcionalidad muy pobre.

La estructura básica de la librería gira en torno a componentes y contenedores.

Los contenedores contienen componentes y son componentes a su vez, de forma que los eventos pueden tratarse tanto en contenedores como en componentes.

AWT es adecuada para interfaces gráficas sencillas, pero no para proyectos más complejos.

Más tarde, cuando apareció Java 2 surgió una librería más robusta, versátil y flexible: **Swing**. AWT aún sigue estando disponible, de hecho se usa por los componentes de Swing.

Principales componentes AWT

NOMBRE DE LA CLASE AWT	UTILIDAD DEL COMPONENTE
Applet	Ventana para una applet que se incluye en una página web.
Button	Crea un botón de acción.
Canvas	Crea un área de trabajo en la que se puede dibujar. Es el único componente AWT que no tiene un equivalente Swing.
Checkbox	Crea una casilla de verificación.
Label	Crea una etiqueta.
Menu	Crea un menú.
ComboBox	Crea una lista desplegable.
List	Crea un cuadro de lista.
Frame	Crea un marco para las ventanas de aplicación.
Dialog	Crea un cuadro de diálogo.
Panel	Crea un área de trabajo que puede contener otros controles o componentes.
PopupMenu	Crea un menú emergente.
RadioButton	Crea un botón de radio.
ScrollBar	Crea una barra de desplazamiento.
ScrollPane	Crea un cuadro de desplazamiento.
TextArea	Crea un área de texto de dos dimensiones.
TextField	Crea un cuadro de texto de una dimensión.
Window	Crea una ventana.

Página oficial de Sun – Oracle sobre AWT (en inglés)

<http://download.oracle.com/javase/6/docs/technotes/guides/awt/>

AWT sigue siendo imprescindible, ya que todos **los componentes Swing se construyen haciendo uso de clases de AWT**. De hecho, como puedes comprobar en el API, todos los componentes Swing, como por ejemplo `JButton` (es la clase Swing que usamos para crear cualquier botón de acción en una ventana), derivan de la clase `JComponent`, que a su vez deriva de la clase AWT `Container`.

Las clases asociadas a cada uno de los componentes AWT se encuentran en el paquete **java.awt**.

Las clases relacionadas con el manejo de eventos en AWT (*Es el modo que tiene una clase de proporcionar notificaciones a los clientes de la clase cuando ocurre algo digno de reseñar en un objeto. El uso más habitual para los eventos se produce*

en las interfaces gráficas; normalmente, las clases que representan controles de la interfaz disponen de eventos que se notifican cuando el usuario interactúa con el control (por ejemplo, hacer clic en un botón) están en el paquete **java.awt.event**.

AWT fue la primera forma de construir las ventanas en Java, pero:

- ✓ limitaba la portabilidad,
- ✓ restringía la funcionalidad y
- ✓ requería demasiados recursos.

AWT está indicado para proyectos muy grandes y de gran complejidad.

- Si
- No

Decíamos más arriba, que las JFC proporcionan soporte para impresión. En el siguiente enlace tienes más información sobre impresión en Java.

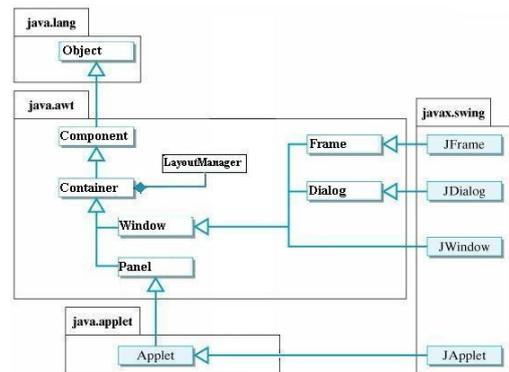
<http://artemisa.unicauca.edu.co/~dparedes/java/jdcbok/render.html>

2.2 Swing.

Cuando se vio que era necesario mejorar las características que ofrecía AWT, distintas empresas empezaron a sacar sus controles propios para mejorar algunas de las características de AWT. Así, Netscape sacó una librería de clases llamada **Internet Foundation Classes** para usar con Java, y eso obligó a Sun (todavía no adquirida por Oracle) a reaccionar para adaptar el lenguaje a las nuevas necesidades.

Se desarrolló en colaboración con Netscape todo el conjunto de componentes Swing que se añadieron a la JFC.

Swing es una librería de Java para la generación del GUI en aplicaciones. Swing se apoya sobre AWT y añade JComponents. La arquitectura de los componentes de Swing facilita la personalización de apariencia y comportamiento, si lo comparamos con los componentes AWT.



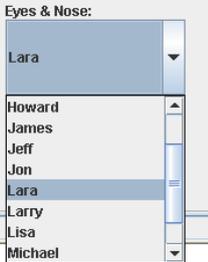
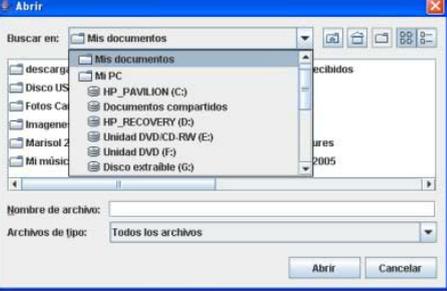
A continuación tienes una tabla con la lista de los controles Swing más habituales. Debes tener en cuenta que las imágenes están obtenidas eligiendo el aspecto (LookAndFeel) multiplataforma propio de Java.

Componentes de Swing

JApplet Ventana para una Applet que se incluye en una página web

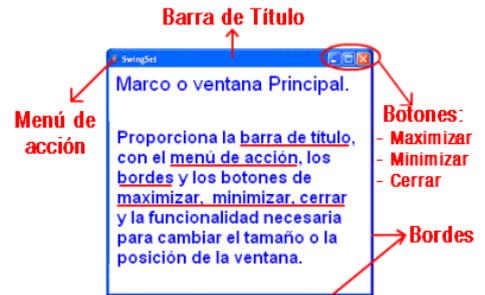
JButton Botón de acción. Para realizar alguna acción o proceso.



JCheckBox	<p>Casilla de verificación. Se usa normalmente para indicar opciones que son independientes, pudiéndose seleccionar algunas, todas o ninguna</p>	
JColorChooser	<p>Panel de controles que permite al usuario seleccionar un color</p>	
JComboBox	<p>Lista desplegable</p>	
JComponent	<p>Clase base para los componentes Swing</p>	
JDesktopPane	<p>Contenedor usado para crear un escritorio, es decir un interfaz para múltiples documentos en la misma aplicación</p>	
JDialog	<p>Clase base para crear un cuadro de diálogo. Los cuadros de diálogo son ventanas que solicitan o proporcionan información al usuario</p>	
JEditorPane	<p>Componente de texto para editar varios tipos de contenido</p>	
JFileChooser	<p>Permite al usuario elegir un archivo para la lectura o escritura, mostrando una ventana que nos permite navegar por los distintos discos y carpetas de nuestro ordenador</p>	

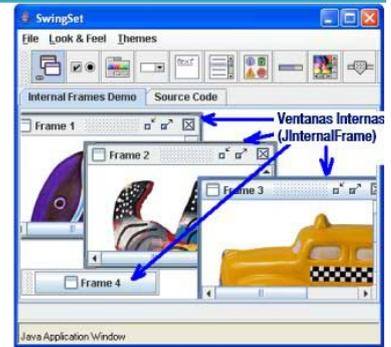
JFrame

Versión extendida de Frame de AWT . Crea un marco para las ventanas de aplicación, dando soporte a los paneles base y otros paneles. Suelen ser las ventanas principales de la aplicación



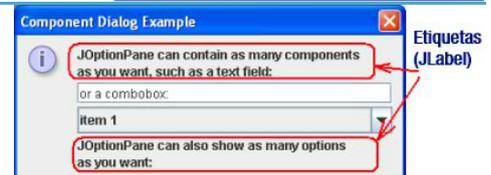
JInternalFrame

Una ventana de peso ligero, que se incluye dentro de un marco JFrame, de forma que no puede sacarse del interior de ese JFrame. Es como un nuevo "documento" en la aplicación principal



JLabel

Una etiqueta que puede contener un texto corto, o una imagen, o ambas cosas

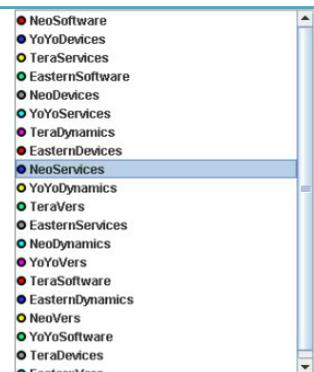


JLayeredPane

Añade capas a un contenedor Swing, de forma que se puedan solapar unos componentes con otros

JList

Un cuadro de lista para seleccionar uno o varios elementos de una lista



JMenu

Un menú de lista desplegable, que incluye JMenuItem y que se visualiza dentro de la barra de menú (JMenuBar)



JMenuBar

Una barra de menú, que aparece generalmente situada debajo de la barra de título de la ventana



JMenuItem

Cada una de las opciones de un menú concreto



JOptionPane	<p>Proporciona implementaciones de cuadros de diálogo emergentes, más o menos estándar y “prefabricados”, que el usuario puede incluir con mucha facilidad en su aplicación</p>	
JPanel	<p>Un área de trabajo que puede contener otros controles o componentes. Es un contenedor Swing de peso ligero</p>	
JPasswordField	<p>Un cuadro de texto donde no se muestran los caracteres que realmente se introducen, sino algún otro (puntos o asteriscos). Se usa para introducir contraseñas de forma que otras personas no puedan ver el valor introducido</p>	
JPopupMenu	<p>Un menú emergente. La imagen del ejemplo está sacada de una aplicación de dibujo</p>	
JPopupMenu.Separator	<p>Separador para un menú emergente específico. Permite hacer grupos con opciones dentro del menú.</p>	<p>Separadores de menú emergente (JPopupMenu.Separator)</p>
JProgressBar	<p>Una barra de progreso, que visualiza gráficamente un valor entero en un intervalo</p>	<p>Barra de progreso (JProgressBar). Se asocia a una tarea para mostrar de una forma gráfica la parte de tarea que ya se ha realizado y la que queda pendiente. Al mismo tiempo sirve para que el usuario compruebe que la tarea progresa, que el ordenador no se ha quedado bloqueado. En este ejemplo se asocia a la carga del texto en el editor.</p>
JRadioButton	<p>Crea un botón de radio. Se usa normalmente para seleccionar una opción de entre varias excluyentes entre sí</p>	
JRadioButtonMenuItem	<p>Botón de opción para un elemento de un menú</p>	
JRootPane	<p>Componente fundamental en la jerarquía del contenedor. Es el panel raíz.</p>	
JScrollBar	<p>Una barra de desplazamiento</p>	

JScrollPane

Panel contenedor que gestiona las barras de desplazamiento verticales y horizontales y las cabeceras de filas y columnas.



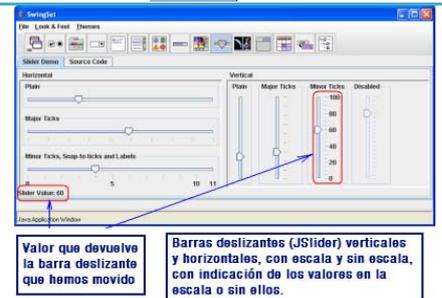
JSeparator

Separador de menú



JSlider

Barra deslizante. Componente que permite seleccionar un valor en un intervalo deslizando un botón



JSplitPane

Panel dividido en dos subpaneles cuyo tamaño relativo puede modificarse pinchando y/o arrastrando la separación entre ambos



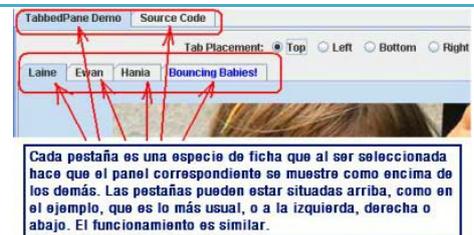
JSpinner

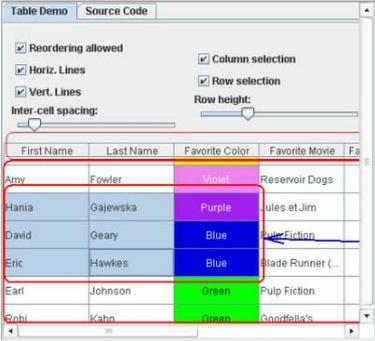
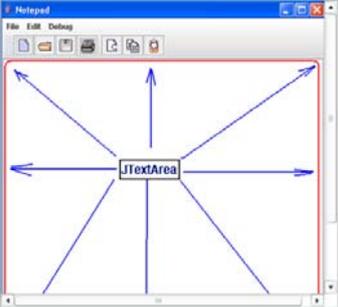
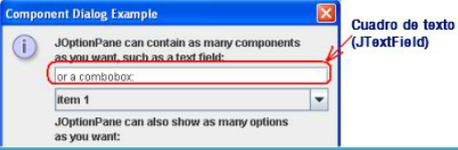
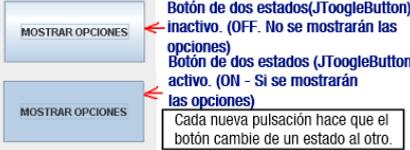
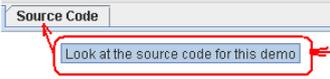
Un cuadro de entrada de una línea que permite seleccionar un número (o el valor de un objeto) de una secuencia ordenada. Normalmente proporciona un par de botones de flecha, para pasar al siguiente o anterior número, (o al siguiente o anterior valor de la secuencia).



JTabbedPane

Panel que contiene múltiples subpaneles, que pueden ser seleccionados haciendo clic sobre la pestaña de cada uno



JTable	Una tabla bidimensional para presentar datos	
JTextArea	Crea un área de texto de dos dimensiones para texto plano	
JTextField	Crea un cuadro de texto de una dimensión	
JTextPane	Componente de texto que se puede marcar con atributos	
JToggleButton	Botón de dos estados (on/off o pulsado/sin pulsar)	
JToolBar	Barra de herramientas, para visualizar controles usados normalmente.	 <p data-bbox="866 1249 1353 1328">Una barra de herramientas proporciona botones con iconos que representan una forma inmediata de ejecutar una determinada acción de la aplicación. Normalmente representan opciones disponibles también en los menús, pero que se usan frecuentemente.</p>
JToolBar.Separator	Separador específico de barras de herramientas	
JToolTip	Una especie de etiqueta emergente, que aparece para visualizar la utilidad de un componente cuando el cursor del ratón pasa por encima de él deteniéndose brevemente, sin necesidad de pulsar nada.	 <p data-bbox="1054 1496 1353 1574">Texto informativo emergente asociado a un elemento, que se visualiza al pasar y parar el ratón por encima del elemento.</p>
JTree	Visualiza un conjunto de datos jerárquicos en forma de árbol	
JViewport	Vista por medio de la cual se ve la información subyacente. Cuando se hace scroll realmente lo que se mueve no es la información, sino el JViewport a través del cual se la mira	
JWindow	Ventana que puede visualizarse en cualquier sitio del escritorio	

Por cada componente AWT (excepto **Canvas**) existe un componente Swing equivalente, cuyo nombre empieza por J, que permite más funcionalidad siendo menos pesado. Así, por ejemplo, para el componente AWT **Button** existe el equivalente Swing **JButton**, que permite como funcionalidad adicional la de crear botones con distintas formas (rectangulares, circulares, etc), incluir imágenes en el botón, tener distintas representaciones para un mismo botón según esté seleccionado, o bajo el cursor, etc.

La razón por la que no existe **JCanvas** es que los paneles de la clase **JPanel** ya soportan todo lo que el componente Canvas de AWT soportaba. No se consideró necesario añadir un componente Swing **JCanvas** por separado.

Algunas características más de Swing, podemos mencionar:

- ✓ Es independiente de la arquitectura (metodología no nativa propia de Java)
- ✓ Proporciona todo lo necesario para la creación de entornos gráficos, tales como diseño de menús, botones, cuadros de texto, manipulación de eventos, etc.
- ✓ **Los componentes Swing no necesitan una ventana propia del sistema operativo cada uno**, sino que son visualizados dentro de la ventana que los contiene mediante métodos gráficos, por lo que requieren bastantes menos recursos.
- ✓ **Las clases Swing están completamente escritas en Java, con lo que la portabilidad es total**, a la vez que no hay obligación de restringir la funcionalidad a los mínimos comunes de todas las plataformas
- ✓ Por ello las clase Swing aportan una considerable gama de funciones que haciendo uso de la funcionalidad básica propia de AWT aumentan las posibilidades de diseño de interfaces gráficas.
- ✓ Debido a sus características, los componentes **AWT** se llaman componentes **“de peso pesado”** por la gran cantidad de recursos del sistema que usan, y los componentes **Swing** se llaman componentes **“de peso ligero”** por no necesitar su propia ventana del sistema operativo y por tanto consumir muchos menos recursos.
- ✓ Aunque todos los componentes Swing derivan de componentes AWT y de hecho se pueden mezclar en una misma aplicación componentes de ambos tipos, se desaconseja hacerlo. Es preferible desarrollar aplicaciones enteramente Swing, que requieren menos recursos y son más portables.

3. Creación de interfaces gráficas de usuario utilizando asistentes y herramientas del entorno integrado.

CASO.

Ana le ha tomado el gusto a hacer programas con la librería Swing de Java y utilizando el IDE NetBeans. Le parece tan fácil que no lo puede creer. Pensaba que sería difícil el uso de los controles de las librerías, pero ha descubierto que es poco menos que cogerlos de la paleta y arrastrarlos hasta el formulario donde quiere situarlos.

- “Pero si esto lo podría hacer hasta mi sobrino pequeño,...” piensa para sí.

Crear aplicaciones que incluyan interfaces gráficas de usuario, con NetBeans, es muy sencillo debido a las facilidades que nos proporcionan sus asistentes y herramientas.

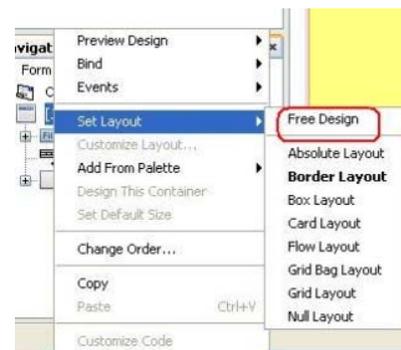
El IDE de programación nos proporciona un diseñador para crear aplicaciones de una manera fácil, sin tener que preocuparnos demasiado del código. Simplemente nos debemos centrar en el funcionamiento de las mismas.

Un programador experimentado, probablemente no utilizará los asistentes, sino que escribirá, casi siempre, todo el código de la aplicación. De este modo, podrá indicar por código la ubicación de los diferentes controles, de su interfaz gráfica, en el contenedor (panel, marco, etc) en el que se ubiquen.

Pero en el caso de programadores novatos, o simplemente si queremos ahorrar tiempo y esfuerzo, tenemos la opción de aprovecharnos de las capacidades que nos brinda un entorno de desarrollo visual para diseñar e implementar formularios gráficos.

Algunas de las características de NetBeans, que ayudan a la generación rápida de aplicaciones, y en particular de interfaces son:

- ✓ Modo de diseño libre (Free Design): permite mover libremente los componentes de interfaz de usuario sobre el panel o marco, sin atenerse a uno de los layouts por defecto.
- ✓ Independencia de plataforma: diseñando de la manera más fácil, es decir, arrastrando y soltando componentes desde la paleta al área de diseño visual, el NetBeans sugiere alineación, posición, y dimensión de los componentes, de manera que se ajusten para cualquier plataforma, y en tiempo de ejecución el resultado sea el óptimo, sin importar el sistema operativo donde se ejecute la aplicación.
- ✓ Soporte de internacionalización. Se pueden internacionalizar las aplicaciones Swing, aportando las traducciones de cadenas de caracteres, imágenes, etc, sin necesidad de tener que reconstruir el proyecto, sin tener que compilarlo según el país al que vaya dirigido. Se puede utilizar esta características empleando ficheros de recursos (ResourceBundle files). En ellos, deberíamos aportar todos los textos visibles, como el texto de etiquetas, campos de texto, botones, etc. NetBeans proporciona una asistente de internacionalización desde el menú de herramientas (Tools).



AUTOEVALUACIÓN.

NetBeans ayuda al programador de modo que pueda concentrarse en implementar la lógica de negocio que se tenga que ejecutar por un evento dado.



Si



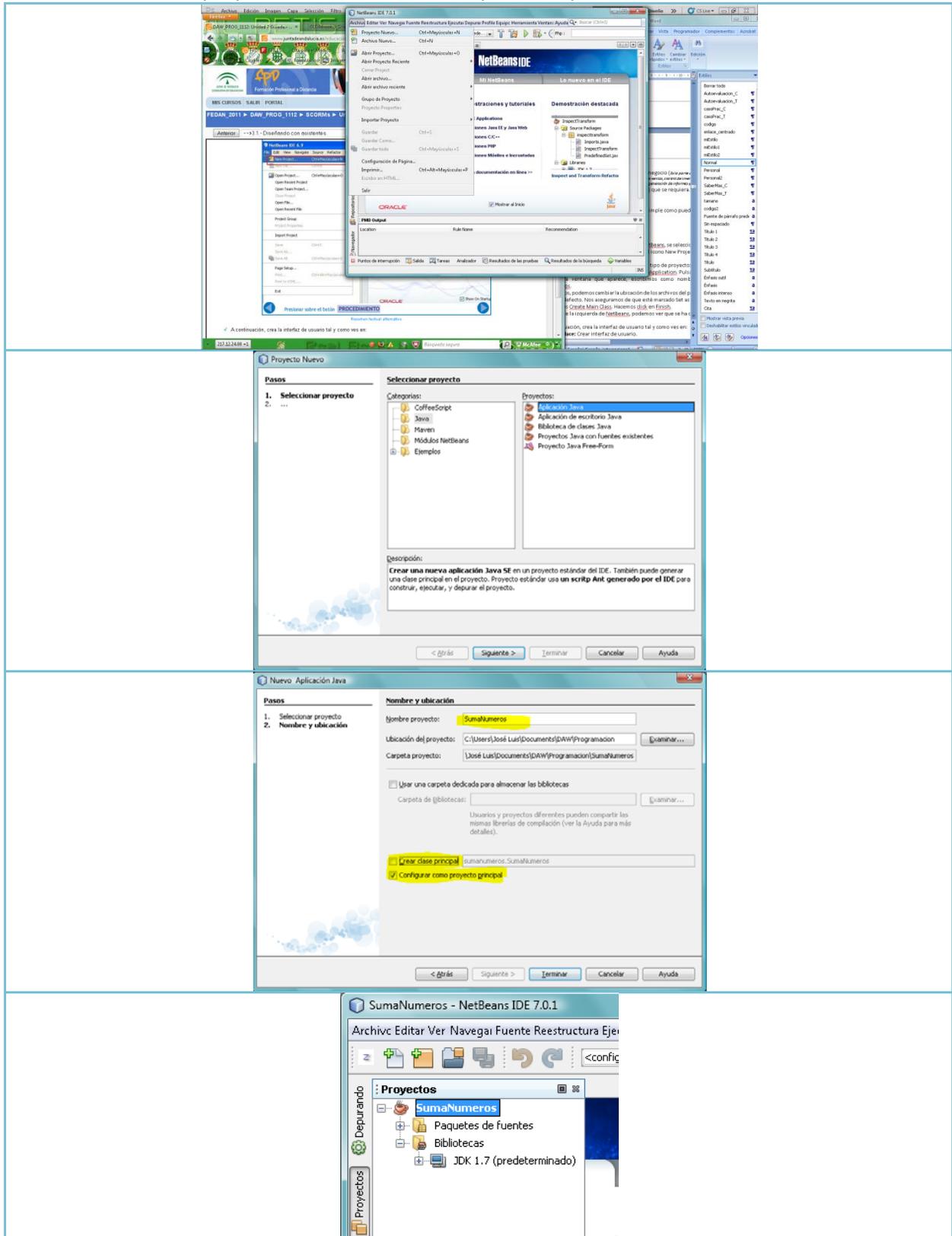
No

3.1 Diseñando con asistentes.

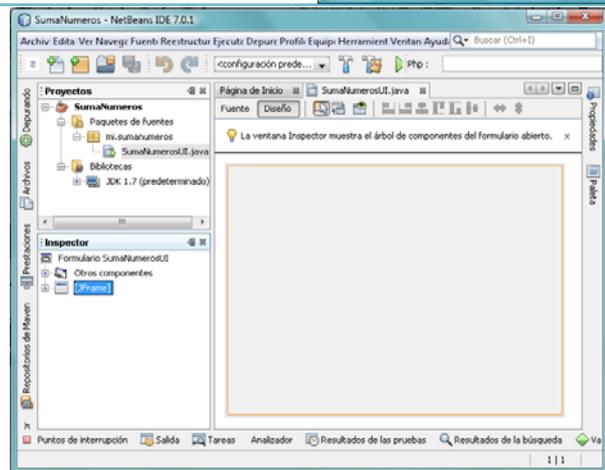
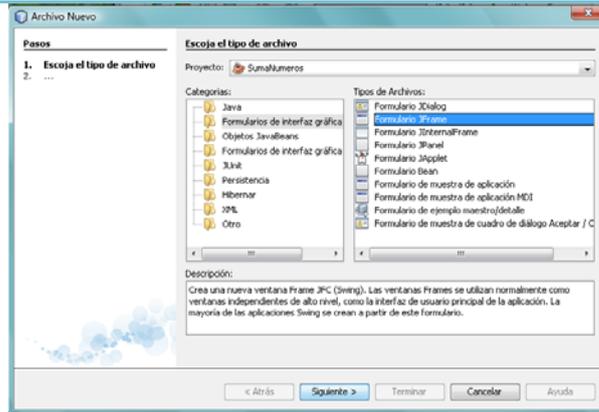
Los pasos para crear y ejecutar aplicaciones, se pueden resumir en:

- ✓ Crear un proyecto.
- ✓ Construir la interfaz con el usuario.

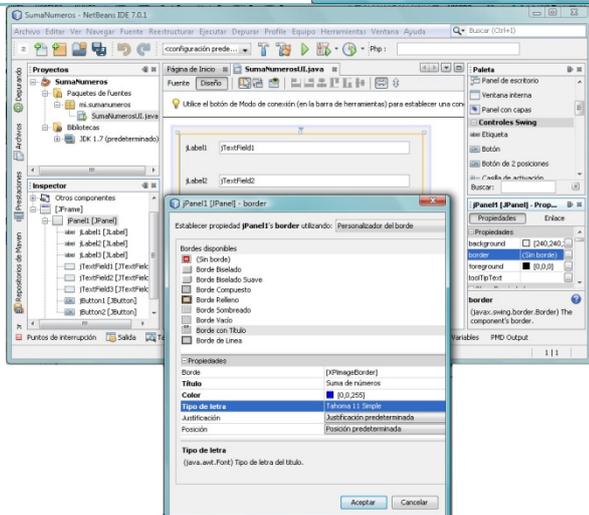
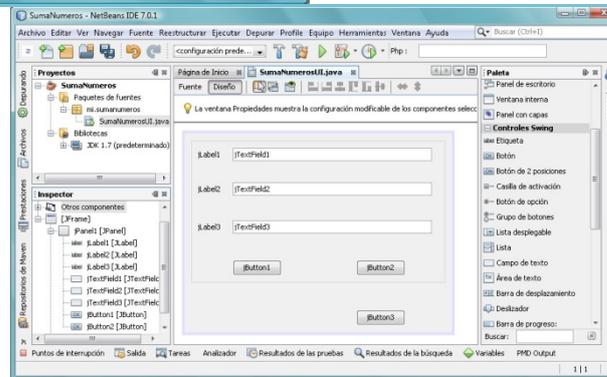
- ✓ Añadir funcionalidad, en base a la lógica de negocio (*Es la parte de la aplicación que contiene el código de las tareas relacionadas con los procesos de un negocio, tales como ventas, control de inventario, contabilidad, etc. Normalmente realizan procesos tales como entradas de datos, consultas a los datos, generación de informes y más específicamente todo el procesamiento que se realiza detrás de la aplicación visible para el usuario*) que se requiera.
 - ✓ Ejecutar el programa.
- Veamos un ejemplo con estos pasos:
- ✓ Primero, crea el proyecto de la manera tan simple como puedes ver en:



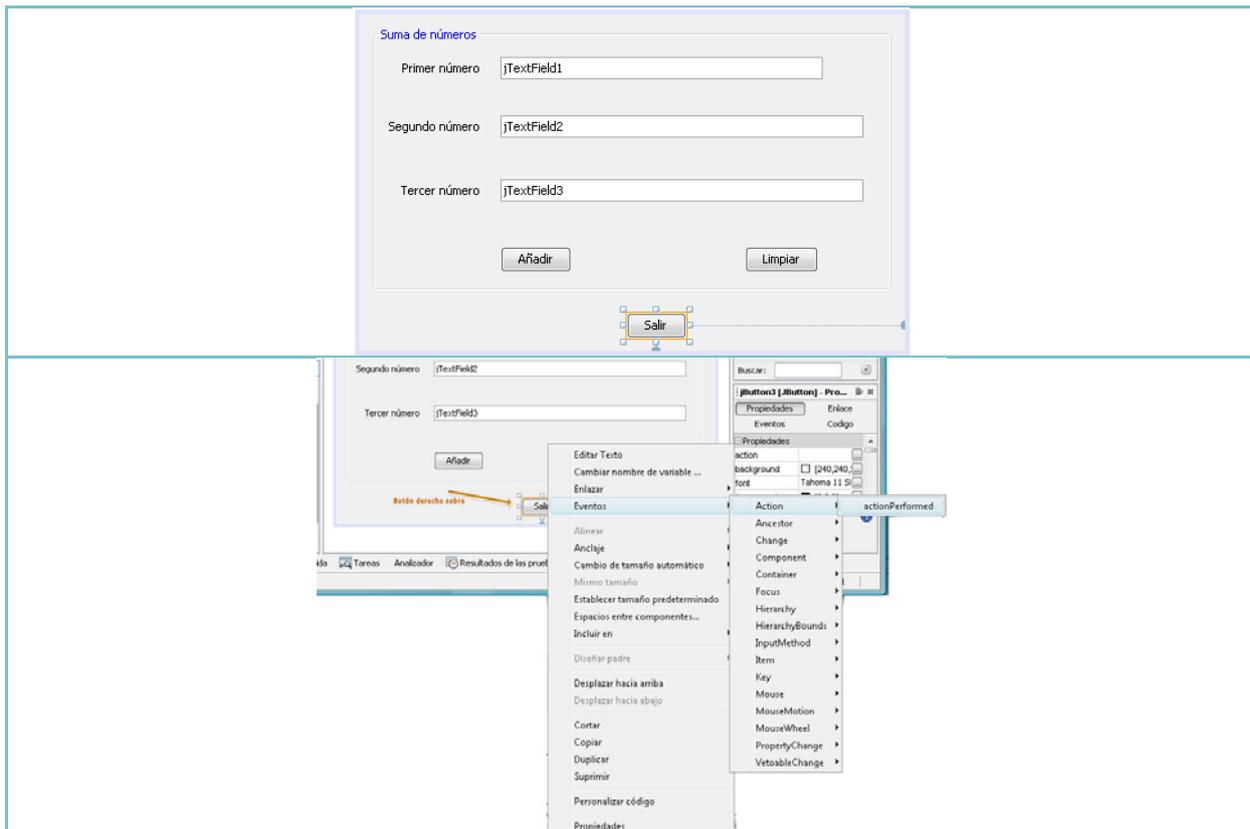
✓ A continuación, crea la interfaz de usuario tal y como ves en:



Necesitamos crear un contenedor Java en el cual dispondremos los otros componentes de interfaz gráfica de usuario necesarios.



Seleccionamos *Borde con título* y escribimos *Suma de Números* en el campo título.



Código que se genera de forma automática, y donde incluiremos lo puesto en negrita.

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    System.exit(0);
}
```

Procederemos igual para el botón *Limpiar* donde pondremos:

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    jTextField1.setText("");
    jTextField2.setText("");
    jTextField3.setText("");
}
```

De igual forma para el botón añadir teclaremos:

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // Primero definimos las variables float
    float num1, num2, result;
    // Convertimos el texto a float
    num1 = Float.parseFloat(jTextField1.getText());
    num2 = Float.parseFloat(jTextField2.getText());
    // Realizamos la suma
    result = num1 + num2;
    // Pasamos el valor del resultado al jTextField3
    // A la misma vez, pasamos el valor de float a string
    jTextField3.setText(String.valueOf(result));
}
```

El IDE, en nuestro caso NetBeans, puede ayudarnos a encontrar la lista de eventos disponibles que nuestros componentes GUI pueden manejar.

Cuando estamos diseñando el interfaz, seleccionando cualquier componente, se puede acceder a la pestaña eventos.

En la pestaña **Events** podemos ver y editar manejadores de eventos asociados al componente GUI activo actualmente.

Cada vez que se presiona un botón, se genera un `ActionEvent` y se pasa al método `actionPerformed` del `listener`, el cual, ejecuta el código que nosotros dispusimos en el manejador de evento para este evento.

Para ser capaz de responder, cada componente GUI interactivo necesita registrar a un oyente de eventos y necesita implementar un manejador de eventos.

Como vemos, el IDE NetBeans se encarga de enganchar el oyente de evento por nosotros, de manera que podamos concentrarnos en implementar la lógica de negocio que se tenga que ejecutar por el evento.

En el siguiente enlace puedes ver, paso a paso, la creación de un proyecto, que utiliza interfaz gráfica de usuario, con NetBeans,

<http://www.udb.edu.sv/Academia/Laboratorios/informatica/Java/guia6Java.pdf>

En este otro enlace también tienes un documento muy interesante, también paso a paso, sobre construcción de interfaces gráficas con NetBeans.

[ftp://soporte.uson.mx/PUBLICO/02_ING.SISTEMAS.DE.INFORMACION/7 Interfaces grficos de usuario con Netbeans ESPANHOL.pdf](ftp://soporte.uson.mx/PUBLICO/02_ING.SISTEMAS.DE.INFORMACION/7%20Interfaces%20graficos%20de%20usuario%20con%20Netbeans%20ESPANHOL.pdf)

En el siguiente vídeo puede ver el primero de una serie de cinco vídeos en los que se realiza una calculadora con la ayuda de los asistentes de NetBeans.

<http://www.youtube.com/watch?v=A9ZX5rWcDOE&feature=related>

4. Eventos.

4.1 Introducción.

CASO.

Ana sabe que entender cómo funciona la programación por eventos es algo relativamente fácil, el problema está en utilizar correctamente los eventos más adecuados en cada momento. Ana tiene muy claro que el evento se asocia a un botón cuando se pulsa, pero José Javier la pone en duda, cuando la llama por teléfono para preguntarle unas dudas y le dice, que él cree, que el evento se produce cuando el botón se suelta. Además, le recuerda que en clase dijeron que al ser enfocado con el ratón, o al accionar una combinación de teclas asociadas, también se pueden producir eventos. Tras hablarlo piensan que realmente no es tan complicado, porque se repiten muchos eventos y si nos paramos a pensarlo, todos ellos son predecibles y bastante lógicos.

¿Qué es un evento?

Es todo hecho que ocurre mientras se ejecuta la aplicación. Normalmente, llamamos evento a cualquier interacción que realiza el usuario con la aplicación, como puede ser:

- ✓ pulsar un botón con el ratón,
- ✓ hacer doble clic,
- ✓ pulsar y arrastrar,
- ✓ pulsar una combinación de teclas en el teclado,
- ✓ pasar el ratón por encima de un componente,
- ✓ salir el puntero de ratón de un componente,
- ✓ abrir una ventana, etc.

¿Qué es la programación guiada por eventos?

Imagina la ventana de cualquier aplicación, por ejemplo la de un procesador de textos. En esa ventana aparecen multitud de elementos gráficos interactivos, de forma que no es posible que el programador haya previsto todas las posibles entradas que se pueden producir por parte del usuario en cada momento.

Con el control de flujo de programa de la **programación imperativa**, el programador tendría que estar continuamente leyendo las entradas (de teclado, o ratón, etc) y comprobar para cada entrada o interacción producida por el usuario, de cual se trata de entre todas las posibles, usando estructuras de flujo condicional (`if then-else`, `switch`) para ejecutar el código conveniente en cada caso. Si piensas que para cada opción del menú, para cada botón o etiqueta, para cada lista desplegable, y por tanto para cada componente de la ventana, incluyendo la propia ventana, habría que comprobar todos y cada uno de los eventos posibles, nos damos cuenta de que las posibilidades son casi infinitas, y desde luego impredecibles. Por tanto, de ese modo es imposible solucionar el problema.

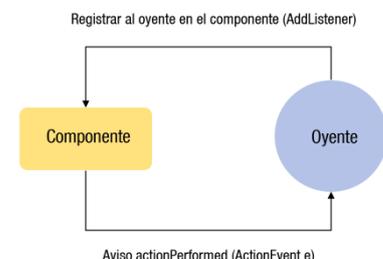
Para abordar el problema de tratar correctamente las interacciones del usuario con la interfaz gráfica de la aplicación hay que cambiar de estrategia, y la **programación guiada por eventos es una buena solución**, veamos cómo funciona el modelo de gestión de eventos.

4.2 Modelo de gestión de eventos.

¿Qué sistema operativo utilizas? ¿Posee un entorno gráfico? Hoy en día, la mayoría de sistemas operativos utilizan interfaces gráficas de usuario. Este tipo de sistemas operativos están **continuamente monitorizando el entorno para capturar y tratar los eventos** que se producen.

El sistema operativo informa de estos eventos a los programas que se están ejecutando y entonces cada programa decide, según lo que se haya programado, qué hace para dar respuesta a esos eventos.

Cada vez que el usuario realiza una determinada acción sobre una aplicación que estamos programando en Java, un clic sobre el ratón, presionar una tecla, etc, se produce un evento que el sistema operativo transmite a Java.



Java crea un objeto de una determinada clase de evento, y este evento se transmite a un determinado método para que lo gestione.

El **modelo de eventos de Java está basado en delegación**, es decir, la responsabilidad de gestionar un evento que ocurre en un objeto fuente la tiene otro objeto **oyente**.

Las **fuentes de eventos** (`event sources`) son objetos que detectan eventos y notifican a los receptores que se han producido dichos eventos. Ejemplos de fuentes:

- ✓ Botón sobre el que se pulsa o pincha con el ratón.
- ✓ Campo de texto que pierde el foco.
- ✓ Campo de texto sobre el que se presiona una tecla.
- ✓ Ventana que se cierra.
- ✓ Etc.

En el apartado anterior de creación de interfaces con ayuda de los asistentes del IDE, vimos lo fácil que es realizar este tipo de programación, ya que el IDE hace muchas cosas, genera código automáticamente por nosotros.

Pero también podríamos hacerlo nosotros todo, si no tuviéramos un IDE como NetBeans, o porque simplemente nos apeteciera hacerlo todo desde código, sin usar asistentes ni diseñadores gráficos.

En este caso, **los pasos a seguir** se pueden resumir en:

1. Crear la clase oyente que implemente la interfaz.
 - Ej. `ActionListener`: pulsar un botón.
2. Implementar en la clase oyente los métodos de la interfaz.
 - Ej. `void actionPerformed(ActionEvent)`.
3. Crear un objeto de la clase oyente y registrarlo como oyente en uno o más componentes gráficos que proporcionen interacción con el usuario.

Observa un ejemplo muy sencillo para ver estos tres pasos:

Creando el programa:

```
public class Ventanilla extends JFrame {
    JLabel etiqueta; // Declarar variables
    JButton botonUno, botonDos;
    JPanel panel;
    public Ventanilla(){
        // Creamos los componentes: una etiqueta,
        // dos botones y un JPanel.
        etiqueta = new JLabel("La etiqueta: ");
        botonUno = new JButton("Uno");
        botonDos = new JButton("Dos");
        panel = new JPanel();
        // Añadimos los componentes al panel
        panel.add(etiqueta);
        panel.add(botonUno);
        panel.add(botonDos);
        // Añadir el panel al Frame
        getContentPane().add(panel);
        // Crear objeto de la clase oyente para cuando se pulse el botón
        OyenteAccion oyenteBoton = new OyenteAccion();
        // Registrar el objeto como oyente en los dos botones
        botonUno.addActionListener(oyenteBoton);
        botonDos.addActionListener(oyenteBoton);
    }
}
// Implementación en la clase oyente
class OyenteAccion implements ActionListener{
    // Cuando se pinche en el botón
    public void actionPerformed(ActionEvent evento){
        // Obtener el botón que disparó el evento
        JButton botón = (JButton) evento.getSource();
        // Escribir en la etiqueta, el botón que se pulsó
        etiqueta.setText("Botón pulsado: " + botón.getText());
    }
}
// Programa principal
public static void main(String args[]){
    // Crear la ventana
    Ventanilla Ventana = new Ventanilla();
    // Establecer el título, el tamaño y hacerla visible
    ventana.setTitle("Título de la ventana");
}
```

```

    ventana.setSize(350,80);
    ventana.setVisible(true);
}
}

```

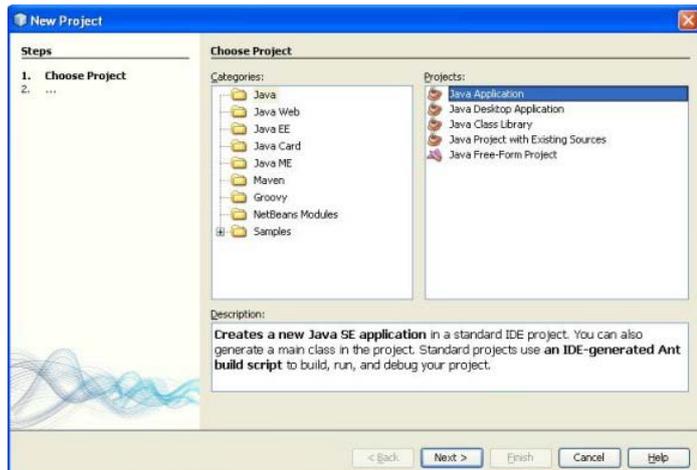
Al ejecutarlo:



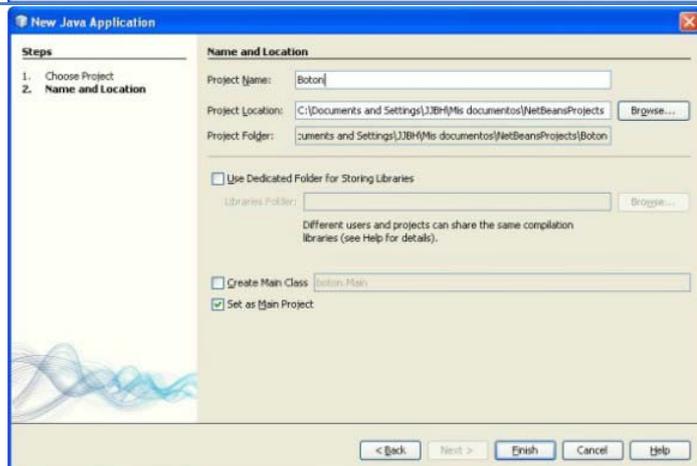
Según pulsemos el botón Uno o Dos, se escribirá el nombre del botón pulsado.

Ejemplo del modelo de gestión de eventos

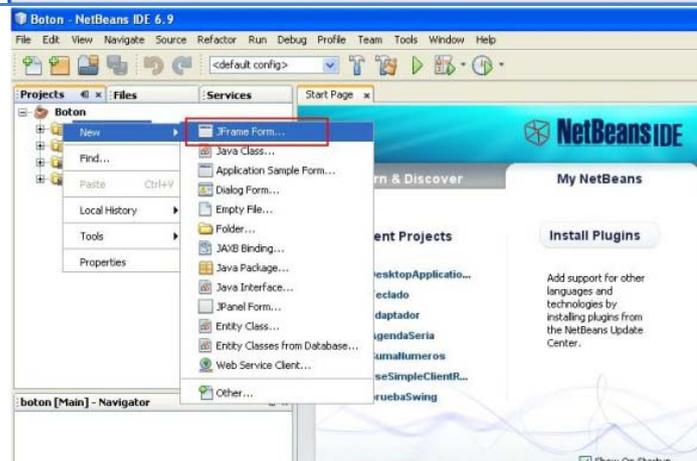
Creamos el proyecto:



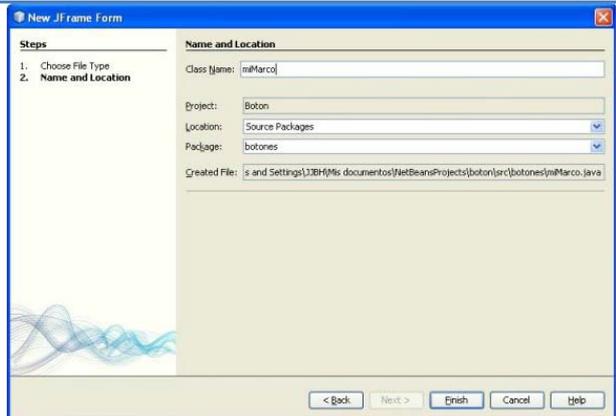
Damos nombre al proyecto



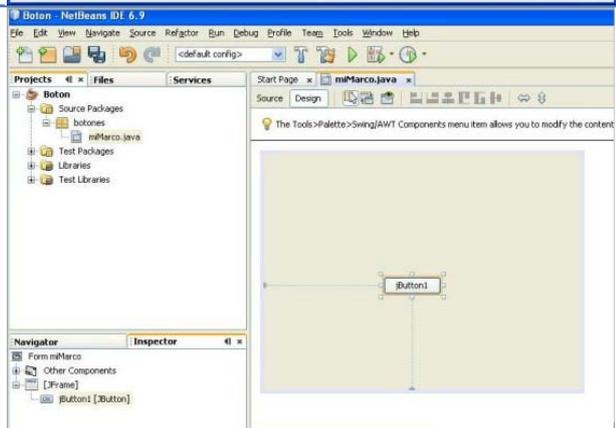
Creamos un nuevo JFrame



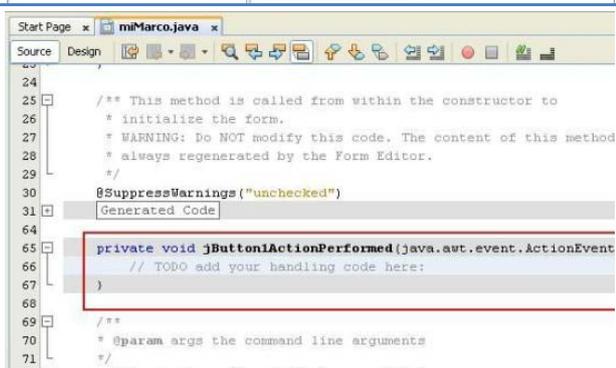
Damos nombre al marco



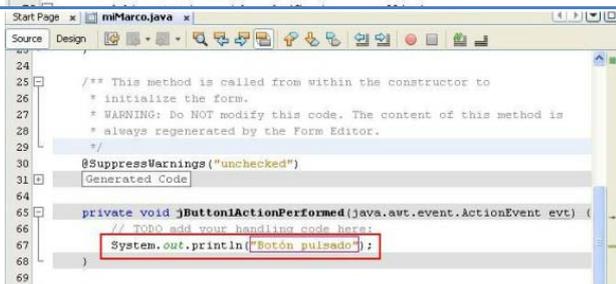
Arrastramos un botón desde la paleta al JFrame



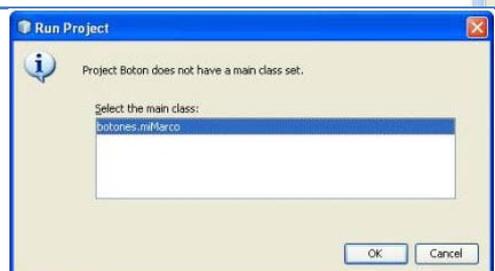
Añadimos la funcionalidad: que escriba por consola botón pulsado. Para ello, hacemos doble click sobre el botón que hemos añadido. Automáticamente NetBeans genera el código que se ve, para que escribamos el código de lo que debe pasar cuando se pulse en el botón.



Añadimos la línea de código para que se escriba por consola

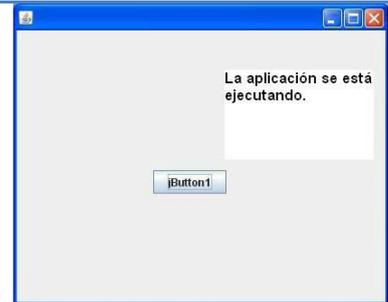


Construimos el proyecto pulsando F11



La aplicación en ejecución:

Hay asociado un oyente de eventos (Listener) apropiado para el tipo de evento que queremos comprobar.
En el ejemplo, hay un ActionListener.



Vemos que NetBeans generó el código señalado.
Se ve que añade un oyente al botón

```

/** This method is called from within the constructor to
 * initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is
 * always regenerated by the Form Editor.
 */
@SuppressWarnings({"unchecked"})
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    jButton1 = new javax.swing.JButton();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

    jButton1.setText("jButton1");
    jButton1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButton1ActionPerformed(evt);
        }
    });

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getCo
    getContentPane().setLayout(layout);
    
```

El oyente es simplemente un programa que permanece activo, en ejecución en segundo plano, de forma paralela a la aplicación.

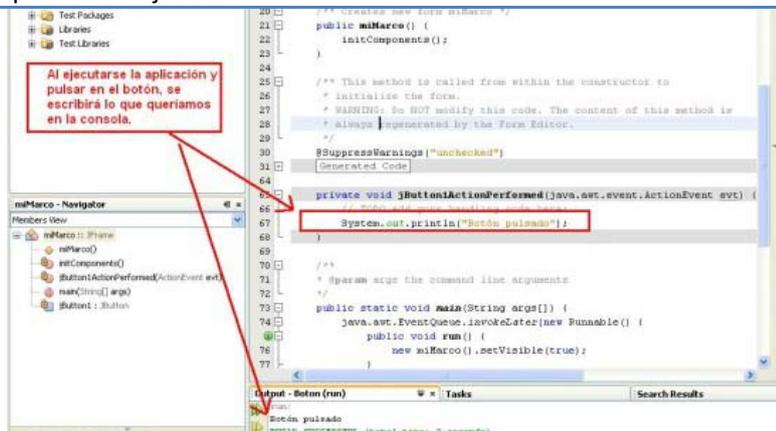
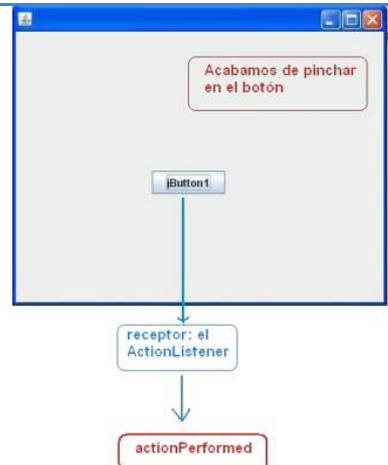
Se encarga exclusivamente de comprobar si se ha producido algún evento del tipo que él sabe escuchar sobre el componente al que se ha añadido

En el caso del botón de acción, el ActionListener se encarga de comprobar si se ha producido algún evento del tipo ActionEvent (evento de acción, como pulsar con el ratón o seleccionar con el atajo de teclado, por ejemplo) sobre el botón.

En el momento que el oyente “oye” o intercepta un evento de ese tipo, lo captura, y pasa el control del flujo de nuestra aplicación al Controlador del componente al que está asociado.

El controlador no es más que el código que el programador ha escrito para que se ejecute cuando se produce exactamente ese evento sobre ese componente. Este código recibe del listener además del control, un parámetro que es un objeto Evento del tipo del evento escuchado por el oyente.

En el caso del botón de acción, se transfiere el control al método actionPerformed() definido por el programador, pasándole como parámetro el objeto evento del tipo ActionEvent que se ha producido. El método actionPerformed() contiene todas las sentencias que deben ejecutarse.



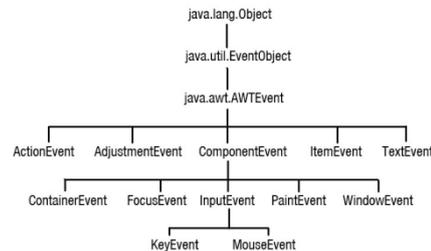
Con la programación guiada por eventos, el programador se concentra en estar continuamente leyendo las entradas de teclado, de ratón, etc, para comprobar cada entrada o interacción producida por el usuario

- Si
- No

4.3 Tipos de eventos.

En la mayor parte de la literatura escrita sobre Java, encontrarás dos tipos básicos de eventos:

- ✓ **Físicos** o de **bajo nivel**: que corresponden a un evento hardware claramente identificable. Por ejemplo, se pulsó una tecla (`KeyStrokeEvent`). Destacar los siguientes:
 - ➔ En componentes: `ComponentEvent`. Indica que un componente se ha movido, cambiado de tamaño o de visibilidad
 - ➔ En contenedores: `ContainerEvent`. Indica que el contenido de un contenedor ha cambiado porque se añadió o eliminó un componente.
 - ➔ En ventanas: `WindowEvent`. Indica que una ventana ha cambiado su estado.
 - ➔ `FocusEvent`, indica que un componente ha obtenido o perdido la entrada del **foco**.
- ✓ **Semánticos** o de mayor nivel de abstracción: se componen de un conjunto de eventos físicos, que se suceden en un determinado orden y tienen un significado más abstracto. Por ejemplo: el usuario elige un elemento de una lista desplegable (`ItemEvent`).
 - ➔ `ActionEvent`, `ItemEvent`, `TextEvent`, `AdjustmentEvent`



Los eventos en Java se organizan en una jerarquía de clases:

- ✓ La clase `java.util.EventObject` es la clase base de todos los eventos en Java.
- ✓ La clase `java.awt.AWTEvent` es la clase base de todos los eventos que se utilizan en la construcción de GUIs.
- ✓ Cada tipo de evento *loqueseaEvent* tiene asociada una interfaz *loqueseaListener* que nos permite definir manejadores de eventos.
- ✓ Con la idea de simplificar la implementación de algunos manejadores de eventos, el paquete `java.awt.event` incluye clases `loqueseaAdapter` que implementan las interfaces *loqueseaListener*.

El evento que se dispara cuando le llega el foco a un botón es un evento de tipo físico.

- Si
- No

4.4 Eventos de teclado.

Los eventos de teclado se generan como respuesta a que el usuario pulsa o libera una tecla mientras un componente tiene el foco de entrada.

KeyListener (oyente de teclas)	
Método	Causa de la invocación
<code>keyPressed (KeyEvent e)</code>	Se ha pulsado una tecla
<code>keyReleased (KeyEvent e)</code>	Se ha liberado una tecla
<code>keyTyped (KeyEvent e)</code>	Se ha pulsado (y a veces soltado) una tecla

KeyEvent (evento de teclas)	
Métodos más usuales	Explicación
<code>char getKeyChar()</code>	Devuelve el carácter asociado con la tecla pulsada

int getKeyCode()	Devuelve el valor entero que representa la tecla pulsada
String getKeyText()	Devuelve un texto que representa el código de la tecla
Object getSource()	Método perteneciente a la clase EventObject. Indica el objeto que produjo el evento

La clase KeyEvent, define muchas constantes así:

- ✓ `KeyEvent.VK_A` especifica la tecla A.
- ✓ `KeyEvent.VK_ESCAPE` especifica la tecla ESCAPE

En la siguiente presentación tienes el código del proyecto que te puedes descargar también a continuación. En él se puede ver un ejemplo del uso eventos. En concreto vemos cómo se están capturando los eventos que se producen al pulsar una tecla y liberarla. El programa escribe en un área de texto las teclas que se oprimen.

Código del oyente de teclado

```
import javax.swing.* ;
import java.awt.event.* ;
/**
 *
 * @author JJBH
 */
// Definimos la clase que hereda de JFrame
public class EscuchaTeclas extends JFrame {
    // Variables para escribir
    private String lineal = "", linea2 = "", linea3 = "";
    private JTextArea areaTexto;

    // Constructor de la clase
    public EscuchaTeclas () {
        // Crear objeto JTextArea
        areaTexto = new JTextArea( 10, 15 );
        areaTexto.setText( "Pulsa cualquier tecla del teclado..." );
        areaTexto.setEnabled( false );

        // Añadir al JFrame el objeto areaTexto
        this.getContentPane().add( areaTexto );

        // Crear el objeto oyente de teclas
        OyenteTeclas oyenteTec = new OyenteTeclas() ;

        // Registrar el oyente en el JFrame
        this.addKeyListener(oyenteTec);
    }

    // Implementar la clase oyente que implemente el interface KeyListener
    class OyenteTeclas implements KeyListener{
        // Gestionar evento de pulsación de cualquier tecla
        public void keyPressed( KeyEvent evento )
        {
            lineal = "Se oprimió tecla: " + evento.getKeyText( evento.getKeyCode() );
            establecerTexto( evento );
        }

        // Gestionar evento de liberación de cualquier tecla
        public void keyReleased( KeyEvent evento )
        {
            lineal = "Se soltó tecla: " + evento.getKeyText( evento.getKeyCode() );
            establecerTexto( evento );
        }

        // manejar evento de pulsación de una tecla de acción
        public void keyTyped( KeyEvent evento )
        {
            lineal = "Se escribió tecla: " + evento.getKeyChar();
            establecerTexto( evento );
        }
    }
}
```

```

// Establecer texto en el componente areaTexto
private void establecerTexto( KeyEvent evento )
{
    // getKeyModifiersText devuelve una cadena que indica
    // el modificador de la tecla, por ejemplo Shift
    String temp = evento.getKeyModifiersText( evento.getModifiers() );

    linea2 = "Esta tecla " + ( evento.isActionKey() ? " " : "no " ) +
        "es una tecla de acción";
    linea3 = "Teclas modificadoras oprimidas: " + ( temp.equals( " " ) ? "ninguna" : temp
);

    // Establecer texto en el componente areaTexto
    areaTexto.setText( linea1 + "\n" + linea2 + "\n" + linea3 + "\n" );
}

public static void main( String args[] )
{
    // Crear objeto y establecer propiedades
    EscuchaTeclas ventana = new EscuchaTeclas();
    ventana.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    ventana.setTitle("Título de la ventana");
    ventana.setSize( 360, 120 );
    ventana.setVisible(true);
}
}

```

4.5 Eventos de ratón.

Similarmente a los eventos de teclado, los eventos del ratón se generan como respuesta a que el usuario pulsa o libera un botón del ratón, o lo mueve sobre un componente.

MouseListener (oyente de ratón)

mousePressed (MouseEvent e)	Se ha pulsado un botón del ratón en un componente
mouseReleased (MouseEvent e)	Se ha liberado un botón del ratón en un componente
mouseClicked (MouseEvent e)	Se ha pulsado y liberado un botón del ratón sobre un componente.
mouseEntered (KeyEvent e)	Se ha entrado (con el puntero del ratón) en un componente.
mouseExited (KeyEvent e)	Se ha salido (con el puntero del ratón) de un componente.

MouseMotionListener (oyente de ratón)

mouseDragged (MouseEvent e)	Se presiona un botón y se arrastra el ratón
mouseMoved (MouseEvent e)	Se mueve el puntero del ratón sobre un componente

MouseWheelListener (oyente de ratón)

MouseWheelMoved (MouseWheelEvent e)	Se mueve la rueda del ratón
--	-----------------------------

En el siguiente proyecto podemos ver una demostración de un formulario con dos botones. Implementamos un oyente `MouseListener` y registramos los dos botones para detectar tres de los cinco eventos del interface.

```

import java.awt.event.* ;

public class miMarcoRaton extends javax.swing.JFrame {

    /** Constructor: crea nuevo marco miMarcoRaton */

```

```

public miMarcoRaton() {
    initComponents();

    // Crear el objeto oyente de ratón
    OyenteRaton oyenteRat = new OyenteRaton() ;

    // Registrar el oyente en el botón de Aceptar
    jButton1.addMouseListener(oyenteRat);
    // Registrar el oyente en el botón de Cancelar
    jButton2.addMouseListener(oyenteRat);
}

// Implementar la clase oyente que implemente el interface MouseListener
// Se deja en blanco el cuerpo de mouseEntered y de mouseExited, ya que
// no nos interesan en este ejemplo. Cuando se desea escuchar algún
// tipo de evento, de deben implementar todos los métodos del interface
// para que la clase no tenga que ser definida como abstracta
class OyenteRaton implements MouseListener{
    // Gestionar evento de pulsación de cualquier tecla
    public void mousePressed(MouseEvent e) {
        escribir("Botón de ratón pulsado", e) ;
    }
    public void mouseReleased(MouseEvent e) {
        escribir("Botón de ratón liberado", e);
    }
    public void mouseEntered(MouseEvent e) {
    }
    public void mouseExited(MouseEvent e) {
    }
    public void mouseClicked(MouseEvent e) {
        escribir("Click en el botón del ratón", e);
    }
}

void escribir(String eventDescription, MouseEvent e) {
    // Escribir en el área de texto la descripción que
    // se recibe como parámetro
    JTextAreal.append(eventDescription + ".\n");

    // Comprobamos cuál de los dos botones es y lo escribimos
    if (e.getComponent().getName().equals(jButton1.getName()) )
        JTextAreal.append("Es el botón Aceptar.\n");
    else
        JTextAreal.append("Es el botón Cancelar.\n");
}

}

/** This method is called from within the constructor to
 * initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is
 * always regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
private void initComponents() {

    jButton1 = new javax.swing.JButton();
    jButton2 = new javax.swing.JButton();
    jScrollPane1 = new javax.swing.JScrollPane();
    JTextAreal = new javax.swing.JTextArea();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

    jButton1.setText("Aceptar");
    jButton1.setName("Aceptar"); // NOI18N

    jButton2.setText("Cancelar");
    jButton2.setName("Cancelar"); // NOI18N

    JTextAreal.setColumns(20);
    JTextAreal.setRows(5);
    jScrollPane1.setViewportView(JTextAreal);

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

```

```

        .addGroup(layout.createSequentialGroup())
            .addGap(80, 80, 80)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
            .addGroup(layout.createSequentialGroup()
                .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE,
402, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
            .addGroup(layout.createSequentialGroup()
                .addComponent(jButton1)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(jButton2)
                .addGap(18, 18, 18)))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addGap(126, 126, 126)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
            .addGroup(layout.createSequentialGroup()
                .addComponent(jButton1)
                .addGap(26, 26, 26))
            .addGroup(layout.createSequentialGroup()
                .addComponent(jButton2)
                .addGap(18, 18, 18)))
                .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 177,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(33, Short.MAX_VALUE))
        );

        pack();
    } // </editor-fold>//GEN-END:initComponents

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new miMarcoRaton().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextArea jTextAreal;
    // End of variables declaration//GEN-END:variables
}

```

Como se ve en el código, se deja en blanco el cuerpo de `mouseEntered` y de `mouseExited`, ya que no nos interesan en este ejemplo. Cuando se desea escuchar algún tipo de evento, de deben implementar todos los métodos del interface para que la clase no tenga que ser definida como abstracta.

Para evitar tener que hacer esto, podemos utilizar adaptadores (es una clase que implemente un oyente o listener, pero no realiza ningún tipo de operación. Se desarrollaron para evitar tener que implementar todos los métodos de un interface, ya que si no se implementan la clase sería abstracta).

Tienes un ejemplo de uso de los adaptadores en el siguiente enlace:

<http://eddi.ith.mx/Curso/Tutoriales/Ozito/post1dot0/ui/eventinnerclasses.html>

En el enlace que ves a continuación, hay también un ejemplo interesante de la programación de eventos del ratón.

<http://www.sc.ehu.es/sbweb/fisica/cursoJava/applets/events/raton.htm>

Cuando el usuario deja de pulsar una tecla se invoca a `keyReleased(KeyEvent e)`.



Si



No

4.6 Creación de controladores de eventos.

A partir del JDK1.4 se introdujo en Java la clase `EventHandler` para soportar oyentes de evento muy sencillos.

La utilidad de estos controladores o manejadores de evento es:

- ✓ Crear oyentes de evento sin tener que incluirlos en una clase propia.
- ✓ Esto aumenta el rendimiento, ya que no “añade” otra clase.

Como inconveniente, destaca la dificultad de construcción: **los errores no se detectan en tiempo de compilación**, sino en tiempo de ejecución.

Por esta razón, es mejor crear controladores de evento con la ayuda de un asistente y documentarlos todo lo posible.

El uso más sencillo de `EventHandler` consiste en instalar un oyente que llama a un método, en el objeto objetivo sin argumentos. En el siguiente ejemplo creamos un `ActionListener` que invoca al método `dibujar` en una instancia de `javax.swing.JFrame`.

```
miBoton.addActionListener(
    (ActionListener)EventHandler.create(ActionListener.class, frame,"dibujar"));
```

Cuando se pulse `miBoton`, se ejecutará la sentencia `frame.dibujar()`. Se obtendría el mismo efecto, con mayor seguridad en tiempo de compilación, definiendo una nueva implementación al interface `ActionListener` y añadiendo una instancia de ello al botón:

```
// Código equivalente empleando una clase interna en lugar de EventHandler.
miBoton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        frame.dibujar();
    }
});
```

Probablemente el uso más típico de `EventHandler` es extraer el valor de una propiedad de la fuente del objeto evento y establecer este valor como el valor de una propiedad del objeto destino. En el siguiente ejemplo se crea un `ActionListener` que establece la propiedad “label” del objeto destino al valor de la propiedad “text” de la fuente (el valor de la propiedad “source”) del evento.

```
EventHandler.create(ActionListener.class, miBoton, "label", "source.text")
```

Esto correspondería a la implementación de la siguiente clase interna:

```
// Código equivalente utilizando una clase interna en vez de EventHandler.
new ActionListener {
    public void actionPerformed(ActionEvent e) {
        miBoton.setLabel(((JTextField)e.getSource()).getText());
    }
}
```

El uso de `EventHandler` tiene como inconveniente, que los errores no se detectan en tiempo de ejecución.



Si



No

5. Generación de programas en entorno gráfico.

CASO.

José Javier va de camino a la clase práctica en la que él, y el resto de la clase, van a probar a hacer sus primeros pasos en programación visual con entornos gráficos, el profesor les explica que al principio parece que todo es muy fácil sobre el papel, pero en realidad crear un proyecto con componentes gráficos no siempre es fácil. Pero el profesor lo tiene claro, hay que empezar por los contenedores que, como su propio nombre indica, se emplean para contener o ubicar al resto de componentes.

En este mismo tema, más arriba, has visto la lista de los principales componentes Swing que se incluyen en la mayoría de las aplicaciones, junto con una breve descripción de su uso, y una imagen que nos da una idea de cuál es su aspecto.

También hemos visto un ejemplo de cómo crear con la ayuda de NetBeans unos sencillos programas. Ahora, vamos a ver con mayor detalle, los componentes más típicos utilizados en los programas en Java y cómo incluirlos dentro de una aplicación.

“Algo sólo es imposible hasta que alguien lo dude y termine probando lo contrario”.

Albert Einstein

5.1 Contenedores.

Por los ejemplos vistos hasta ahora, ya te habrás dado cuenta de la necesidad de que cada aplicación “contenga” de alguna forma esos componentes. ¿Qué componentes se usan para contener a los demás?

En Swing esa función la desempeñan un grupo de componentes llamados **contenedores** Swing.

Existen dos tipos de elementos contenedores:

- ✓ **Contenedores de alto nivel** o “peso pesado”.
 - ✓ Marcos: `JFrame` y `JDialog` para aplicaciones
 - ✓ `JApplet`, para applets (es un componente de una aplicación que se ejecuta en el contexto de otro programa, por ejemplo un navegador web. El applet debe ejecutarse en un contenedor, que lo proporciona un programa anfitrión, mediante un plugin, o en aplicaciones como teléfonos móviles que soportan el modelo de programación por applets).
- ✓ **Contenedores de bajo nivel** o “peso ligero”. Son los paneles: `JRootPane` y `JPanel`.

Cualquier aplicación, con interfaz gráfico de usuario típica, comienza con la apertura de una ventana principal, que suele contener la barra de título, los botones de minimizar, maximizar/restaurar y cerrar, y unos bordes que delimitan su tamaño.

Esa ventana constituye un marco dentro del cual se van colocando el resto de componentes que necesita el programador: menú, barras de herramientas, barra de estado, botones, casillas de verificación, cuadros de texto, etc.

Esa ventana principal o marco sería el contenedor de alto nivel de la aplicación.

Toda aplicación de interfaz gráfica de usuario Java tiene, al menos, un contenedor de alto nivel.

Los contenedores de alto nivel extienden directamente a una clase similar de AWT, es decir, `JFrame` extiende de `Frame`. Es decir, realmente necesitan crear una ventana del sistema operativo independiente para cada uno de ellos.

Los demás componentes de la aplicación no tienen su propia ventana del sistema operativo, sino que se dibujan en su objeto contenedor.

En los ejemplos anteriores del tema, hemos visto que podemos añadir un `JFrame` desde el diseñador de NetBeans, o bien escribiéndolo directamente por código. De igual forma para los componentes que añadamos sobre él.

Te recomendamos que mires la siguiente web para ver información sobre `JFrame` y `JDialog`.

http://chuwiki.chuidiang.org/index.php?title=JFrame_y_JDialog

Cualquier componente de gráfico de una aplicación Java necesita tener su propia ventana del sistema operativo.



Si



No

5.2 Cerrar la aplicación.

Cuando quieres terminar la ejecución de un programa, ¿qué sueles hacer? Pues normalmente pinchar en el icono de cierre de la ventana de la aplicación.

En Swing, una cosa es cerrar una ventana, y otra es que esa ventana deje de existir completamente, o cerrar la aplicación completamente.

- ✓ **Se puede hacer que una ventana no esté visible**, y sin embargo que ésta siga existiendo y ocupando memoria para todos sus componentes, usando **el método** `setVisible(false)`. En este caso bastaría ejecutar para el `JFrame` el método `setVisible(true)` para volver a ver la ventana con todos sus elementos.
- ✓ **Si queremos cerrar la aplicación**, es decir, que no sólo se destruya la ventana en la que se mostraba, sino que se destruyan y liberen todos los recursos (memoria y CPU) que esa aplicación tenía reservados, tenemos que invocar al método `System.exit(0)`
- ✓ **También se puede invocar para la ventana `JFrame` al método `dispose()`**, heredado de la clase **Window**, que no requiere ningún argumento, **y que borra todos los recursos de pantalla usados por esta ventana y por sus componentes, así como cualquier otra ventana que se haya abierto como hija de esta** (dependiente de esta). Cualquier memoria que ocupara esta ventana y sus componentes se libera y se devuelve al sistema operativo, y tanto la ventana como sus componentes se marcan como “no representables”. Y sin embargo, el objeto ventana sigue existiendo, y podría ser reconstruido invocando al método `pack()` o la método `show()`, aunque deberían construir de nuevo toda la ventana.

Las ventanas `JFrame` de Swing permiten establecer una operación de cierre por defecto con el método `setDefaultCloseOperation()`, definido en la clase `JFrame`.

¿Cómo se le indica al método el modo de cerrar la ventana?

Los valores que se le pueden pasar como parámetros a este método son una serie de constantes de clase:

- ✓ `DO_NOTHING_ON_CLOSE`: **No hace nada**, necesita que el programa maneje la operación en el método `windowClosing()` de un objeto `WindowListener` registrado para la ventana.
- ✓ `HIDE_ON_CLOSE`: **Oculto** de ser mostrado en la pantalla pero no destruye el marco o ventana después de invocar cualquier objeto `WindowListener` registrado.
- ✓ `DISPOSE_ON_CLOSE`: **Oculto y termina (destruye) automáticamente el marco o ventana** después de invocar cualquier objeto `WindowListener` registrado.
- ✓ `EXIT_ON_CLOSE`: Sale de la aplicación usando el método `System.exit(0)`. Al estar definida en `JFrame`, se puede usar con aplicaciones, pero no con applets.

System.exit(0) oculta la ventana pero no libera los recursos de CPU y memoria que la aplicación tiene reservados.



Si



No

5.3 Organizadores de contenedores: layout managers.

Los layout managers son fundamentales en la creación de interfaces de usuario, ya que determinan las posiciones de los controles en un contenedor.

En lenguajes de programación para una única plataforma, el problema sería menor porque el aspecto sería fácilmente controlable. Sin embargo, dado que Java está orientado a la portabilidad del código, éste es uno de los aspectos más complejos de la creación de interfaces, ya que las medidas y posiciones dependen de la máquina en concreto.

En algunos entornos los componentes se colocan con coordenadas absolutas. En Java se desaconseja esa práctica porque en la mayoría de casos es imposible prever el tamaño de un componente.

Por tanto, en su lugar, se usan organizadores o también llamados **administradores de diseño** o **layout managers** o **gestores de distribución** que permiten colocar y maquetar de forma independiente de las coordenadas.

Debemos hacer un buen diseño de la interfaz gráfica, y así tenemos que elegir el mejor gestor de distribución para cada uno de los contenedores o paneles de nuestra ventana.

Esto podemos conseguirlo con el método `setLayout()`, al que se le pasa como argumento un objeto del tipo de `Layout` que se quiere establecer.

En NetBeans, una vez insertado un `JFrame`, si nos situamos sobre él y pulsamos botón derecho, se puede ver, como muestra la imagen, que aparece un menú, el cual nos permite elegir el `layout` que queramos.

En la siguiente web puedes ver gráficamente los distintos layouts

<http://download.oracle.com/javase/tutorial/uiswing/layout/visual.html>

<http://casidiablo.net/codigo-java-flowlayout-borderlayout-gridlayout/>

A continuación se muestra el uso de **FlowLayout** para posicionar varios botones.

Alignment: LEFT Horizontal Gap: 5 Vertical Gap: 5	Con los valores por defecto. Todos los componentes de la ventana se alinean a la izquierda unos seguidos de otros, siempre que encuentren espacio para ser situados.	
Alignment: CENTER Horizontal Gap: 25 Vertical Gap: 25	Alineación centrada de los componentes aumentando el espacio horizontal y vertical entre los mismos	
Alignment: LEFT Horizontal Gap: 5 Vertical Gap: 5	Con las mismas propiedades del anterior y el mismo Layout, pero aumentando el ancho de la ventana para que quepan más controles. Todos los componentes de la ventana se alinean a la izquierda y todos en la misma fila	
Alignment: RIGHT Horizontal Gap: 25 Vertical Gap: 25	Alineación a la derecha de los componentes con un aumento del espacio horizontal y vertical entre los mismos	
Alignment: LEFT Horizontal Gap: 25 Vertical Gap: 25	Partiendo de la ventana con el ancho inicial, se aumenta el espacio horizontal y vertical entre los controles (botones en este caso), quedando distribuidos de este modo...	

Cuando programamos en Java es aconsejable establecer coordenadas absolutas, siempre que sea posible, en nuestros componentes.



Si



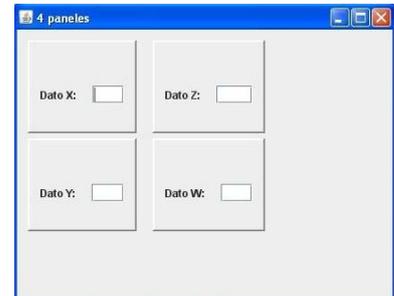
No

5.4 Contenedor ligero: JPanel.

Es la clase utilizada como contenedor genérico para agrupar componentes.

Normalmente cuando una ventana de una aplicación con interfaz gráfica cualquiera presenta varios componentes, para hacerla más atractiva y ordenada al usuario se suele hacer lo siguiente:

- ✓ Crear un marco, un `JFrame`.
- ✓ Para organizar mejor el espacio en la ventana, añadiremos varios paneles, de tipo `JPanel`. (Uno para introducir los datos de entrada, otro para mostrar los resultados y un tercero como zona de notificación de errores.)
- ✓ Cada uno de esos paneles estará delimitado por un borde que incluirá un título. Para ello se usan las clases disponibles en `BorderFactory` (`BevelBorder`, `CompoundBorder`, `EmptyBorder`, `EtchedBorder`, `LineBorder`, `LoweredBevelBorder`, `MatteBorder` y `TitledBorder`) que nos da un surtido más o menos amplio de tipos de bordes a elegir.
- ✓ En cada uno de esos paneles se incluyen las etiquetas y cuadros de texto que se necesitan para mostrar la información adecuadamente.



Con NetBeans es tan fácil como arrastrar tantos controles `JPanel` de la paleta hasta el `JFrame`. Por código, también es muy sencillo, por ejemplo podríamos crear un panel rojo, darle sus características y añadirlo al `JFrame` del siguiente modo:

```
...
JPanel panelRojo = new JPanel();
panelRojo.setBackground(Color.RED);
panelRojo.setSize(300,300);
// Se crea una ventana
JFrame ventana=new JFrame("Prueba en rojo");
ventana.setLocation(100,100);
ventana.setVisible(true);
// Se coloca el JPanel en el content pane
Container contentPane=ventana.getContentPane();
contentPane.add(panelRojo);
...
```

Cuando en Sun desarrollaron Java, los diseñadores de Swing, por alguna circunstancia, determinaron que para algunas funciones, como añadir un `JComponent`, los programadores no pudiéramos usar `JFrame.add`, sino que en lugar de ello, primeramente, tuviéramos que obtener el objeto `Container` asociado con `JFrame.getContentPane()`, y añadirlo.

Sun se dio cuenta del error y ahora permite utilizar `JFrame.add`, desde Java 1.5 en adelante. Sin embargo, podemos tener el problema de que un código desarrollado y ejecutado correctamente en 1.5 falle en máquinas que tengan instalado 1.4 o anteriores.

En la siguiente web puedes ver algo más sobre paneles.

<http://members.fortunecity.es/mastermdei/usuarios.tripode.es/panel.html>

5.5 Etiquetas y campos de texto.

Los **cuadros de texto** Swing vienen implementados en Java por la clase `JTextField`.

Para insertar un campo de texto, el procedimiento es tan fácil como: **seleccionar el botón correspondiente a `JTextField` en la paleta de componentes**, en el diseñador,



y **pinchar sobre el área de diseño** encima del panel en el que queremos situar ese campo de texto. El tamaño y el lugar en el que se sitúe, dependerá del `Layout` elegido para ese panel.

El componente Swing etiqueta `JLabel`, se utiliza para crear etiquetas de modo que podamos insertarlas en un marco o un panel para visualizar un texto estático, que no puede editar el usuario.

Los constructores son:

- ✓ `JLabel()`. Crea un objeto `JLabel` sin nombre y sin ninguna imagen asociada.
- ✓ `JLabel(Icon imagen)`. Crea un objeto sin nombre con una imagen asociada.
- ✓ `JLabel(Icon imagen, int alineacionHorizontal)`. Crea una etiqueta con la imagen especificada y la centra en horizontal.
- ✓ `JLabel(String texto)`. Crea una etiqueta con el texto especificado.
- ✓ `JLabel(String texto, Icon icono, int alineacionHorizontal)`. Crea una etiqueta con el texto y la imagen especificada y alineada horizontalmente.
- ✓ `JLabel(String texto, int alineacionHorizontal)`. Crea una etiqueta con el texto especificado y alineado horizontalmente.

Propiedades asociadas a `JTextField`, y los métodos que permiten modificarlas

Propiedad	Métodos asociados	Descripción
Background	<code>setBackground()</code> <code>getBackground()</code>	Establece el color del fondo del cuadro de texto.
Border	<code>setBorder()</code>	Permite seleccionar las características del borde que tiene el cuadro de texto para delimitarlo
Editable	<code>setEditable()</code>	Establece si se va a poder modificar el texto del campo de texto por el usuario de la aplicación. Aunque se establezca su valor a <code>false</code> , el programa sí que podrá modificar ese texto.
Font	<code>setFont()</code> . <code>getFont()</code>	Establece las propiedades de la fuente (tipo de letra, estilo y tamaño del texto)
Foreground	<code>setForeground()</code>	Establece el color del texto.
HorizontalAlignment	<code>setHorizontalAlignment()</code>	Establece la justificación del texto dentro del cuadro de texto (Justificado a la izquierda, centrado o a la derecha).
Text	<code>setText()</code> <code>getText()</code>	Establece el texto que contiene el cuadro de texto. También es posible mezclar texto de distintos colores y fuentes, usando etiquetas HTML como parte del texto. Para más detalles sobre el uso de HTML en el texto de los componentes, debes mirar la sección Using HTML in Swing Components del tutorial de Java, disponible en la documentación del JDK.
ToolTipText	<code>setToolTipText()</code> <code>getToolTipText()</code>	Añade un comentario de ayuda flotante, de forma que se despliega al detener el cursor encima del componente.
Enabled	<code>setEnabled()</code>	Establece si el componente va a estar activo o no. Un componente que no está activo no puede responder a las entradas del usuario. Algunos componentes, como <code>JTextField</code> , modifican su representación visual para dejar claro que no están activos, mostrando un color gris atenuado.
MaximunSize	<code>setMaximumSize()</code>	Establece el tamaño máximo para el

	getMaximumSize()	componente.
MinimunSize	setMinimumSize() getMinimumSize()	Establece el tamaño mínimo para el componente.
PreferredSize	setPreferredSize() getPreferredSize()	Establece el tamaño preferido para el componente. El valor preferido se tendrá en cuenta a la hora de redimensionar la ventana, según el Layout que se haya usado, para procurar que el aspecto sea siempre el mejor posible, pero no hay garantías de que se respete siempre.
Opaque	setOpaque()	Establece si el componente va a ser opaco, o si por el contrario va a permitir que se vean otros componentes que puedan estar debajo de él, como si fuera un cristal transparente.

En el siguiente enlace puedes ver cómo usar DecimalFormat para presentar un número en un JTextField o recoger el texto del JTextField y reconstruir el número.

<http://chuwiki.chuidiang.org/index.php?title=DecimalFormat>

Un componente JLabel permite al usuario de la aplicación en ejecución cambiar el texto que muestra dicho componente.



5.6 Botones.

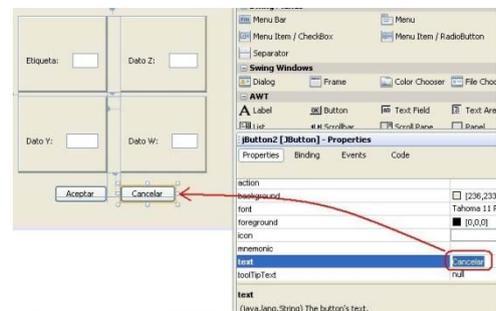
Ya has podido comprobar que prácticamente todas las aplicaciones incluyen botones que al ser pulsados efectúan alguna acción: hacer un cálculo, dar de alta un libro, aceptar modificaciones, etc.

Estos botones se denominan **botones de acción**, precisamente porque realizan una acción cuando se pulsan. En Swing, la clase que los implementa en Java es la `JButton`.

Los principales métodos son:

- ✓ `void setText(String)`: Asigna el texto al botón.
- ✓ `String getText()`: Recoge el texto.

Hay un tipo especial de botones, que se comportan como **interruptores de dos posiciones** o estados (pulsados-on, no pulsados-off). Esos botones especiales se denominan botones on/off o `JToggleButton`.



A continuación puedes ver un par de enlaces: una en el que se diseña una interfaz gráfica de usuario sencilla, con los controles que hemos visto hasta ahora, y el segundo enlace, en inglés, un ejemplo para añadir un botón en Java por código.

http://geocities.ws/wlopezm/articulos/gui_java.pdf

<http://www.apl.jhu.edu/~hall/java/Swing-Tutorial/Swing-Tutorial-JButton.html>

5.7 Casillas de verificación y botones de radio.

Las casillas de verificación en Swing están implementadas para Java por la clase `JCheckBox`, y los botones de radio o de opción por la clase `JRadioButton`. Los grupos de botones, por la clase `ButtonGroup`.

La funcionalidad de ambos componentes es en realidad la misma.

- ✓ Ambos tienen dos “estados”: Seleccionados o no seleccionados. (marcados o no marcados)
- ✓ Ambos se marcan o desmarcan usando el método `setSelected(boolean estado)`, que establece el valor para su propiedad `selected`. (El estado toma el valor `true` para seleccionado y `false` para no seleccionado)
- ✓ A ambos le podemos preguntar si están seleccionados o no, mediante el método `isSelected()`, que devuelve `true` si el componente está seleccionado y `false` si no lo está.
- ✓ Para ambos podemos asociar un icono distinto para el estado de seleccionado y el de no seleccionado.



`JCheckBox` pueden usarse en menús mediante la clase `JCheckBoxMenuItem`.

`JButtonGroup` permite agrupar una serie de casillas de verificación (`JRadioButton`), de entre las que sólo puede seleccionarse una. Marcar una de las casillas implica que el resto sean desmarcadas automáticamente. La forma de hacerlo consiste en añadir un `JButtonGroup` y luego, agregarle los botones.

Cuando en un contenedor aparezcan agrupados varios botones de radio (o de opción), entenderemos que no son opciones independientes, sino que sólo uno de ellos podrá estar activo en cada momento, y necesariamente uno debe estar activo. Por tanto en ese contexto entendemos que son opciones excluyentes entre sí.

En el siguiente enlace puedes ver un vídeo-tutorial para crear un ejemplo básico de Java con interfaz gráfica.

http://www.youtube.com/watch?v=J0u_yf5CZNE&feature=related

Los componentes radiobotones y las casillas de verificación tienen ambos dos estados: seleccionado y no seleccionado.



Si



No

5.8 Listas.

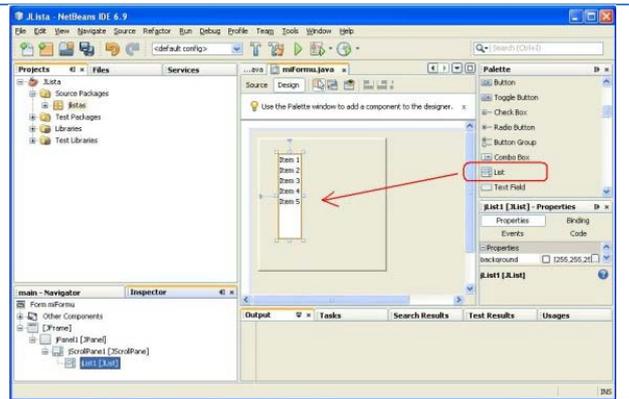
En casi todos los programas, nos encontramos con que se pide al usuario que introduzca un dato, y además un dato de entre una lista de valores posibles, no un dato cualquiera.

La clase `JList` constituye un componente lista sobre el que se puede ver y seleccionar uno o varios elementos a la vez. En caso de haber más elementos de los que se pueden visualizar, es posible utilizar un componente `JScrollPane` para que aparezcan barras de desplazamiento.

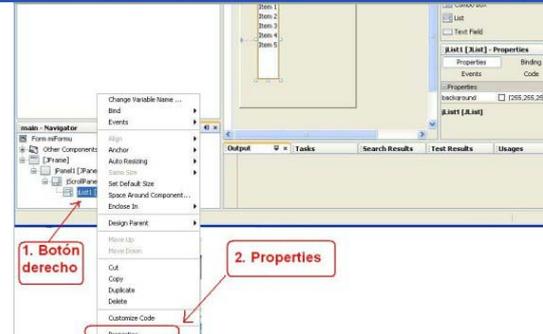


Para añadir un `JList` en NetBeans procederemos como podemos observar en la siguiente página:

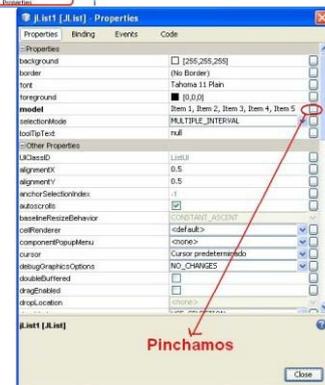
Arrastrar el control **List** de la paleta hasta el panel donde queremos situarlo.
 Por defecto vemos que lleva 5 elementos:
 item1 hasta item 5.



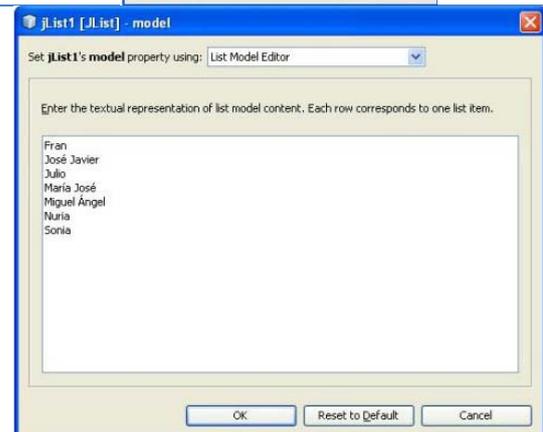
Una vez insertado hacemos botón derecho sobre el componente y seleccionamos las propiedades.



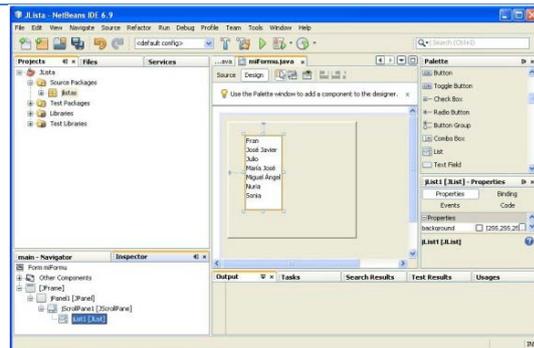
Pinchamos en el botoncillo que hay a la derecha de la propiedad model.



Podemos introducir los elementos que queremos que aparezcan en la lista



Al aceptar vemos que en la lista aparecen los datos que acabamos de introducir



En los componentes `JList`, un modelo `ListModel` representa **los contenidos de la lista**. Esto significa que los datos de la lista se guardan en una estructura de datos independiente, denominada modelo de la lista. Es fácil mostrar en una lista los elementos de un vector o array de objetos, usando un constructor de `JList` que cree una instancia de `ListModel` automáticamente a partir de ese vector. Vemos a continuación un ejemplo para crear una lista `JList` que muestra las cadenas contenidas en el vector `info[]`:

```
String[] info = {"Pato", "Loro", "Perro", "Cuervo"};
JList listaDatos = new JList(info);
/* El valor de la propiedad model de JList es un objeto que proporciona una visión
de sólo lectura del vector info[]. El método getModel() permite recoger ese modelo
en forma de Vector de objetos, y utilizar con los métodos de la clase Vector, como
getElementAt(i), que proporciona el elemento de la posición i del Vector. */
for (int i = 0; i < listaDatos.getModel().getSize(); i++) {
    System.out.println(listaDatos.getModel().getElementAt(i));
}
```

A continuación puedes ver cómo crear un `JList`, paso a paso, en el enlace siguiente.

<http://jc-mouse.blogspot.com/2010/02/anadireliminar-elementos-de-jlist-en.html>

En el enlace que tienes a continuación puedes ver una presentación, que aunque con audio en inglés, es lo suficientemente explicativa para entenderla.

<http://bits.netbeans.org/media/quickstart-gui-align.swf>

Resumen: Se visualiza el entorno NetBeans con un formulario diseñado con varios controles: dos paneles, uno arriba y otro abajo. En el panel de arriba hay cuatro campos de texto con sus respectivas etiquetas y el panel de abajo aparece vacío. Se hace un click en la etiqueta que hay más hacia arriba a la izquierda y entonces se hace click en el botón de alinear a la derecha, que aparece en la barra de herramientas que hay sobre el formulario. Se repite lo mismo para las otras etiquetas. Tras eso, se ve ahora cómo cambiar otro aspecto, esta vez de los campos de texto. Se selecciona el de abajo a la izquierda, y se hace click con el botón derecho del ratón; en el menú contextual se selecciona Autoresizing Horizontal. De este modo, en tiempo de ejecución, ese control se podrá cambiar de tamaño horizontalmente. Se añade ahora una etiqueta más, arrastrándola desde la paleta de componentes de la derecha de la pantalla.

Ahora se añade un control combobox a la derecha de la etiqueta añadida.

Se añade ahora una etiqueta en el panel inferior, y después un campo de texto a su derecha, que se redimensiona arrastrándolo desde su borde derecho. También se añade finalmente una lista.

5.8.1 Listas (II).

Cuando trabajamos con un componente `JList`, podemos seleccionar un único elemento, o varios elementos a la vez, que a su vez pueden estar contiguos o no contiguos. La posibilidad de hacerlo de una u otra manera la establecemos con la propiedad `selectionMode`.

Los valores posibles para la propiedad `selectionMode` para cada una de esas opciones son las siguientes constantes de clase del interface `ListSelectionMode`:

- ✓ `MULTIPLE_INTERVAL_SELECTION`: Es el valor por defecto. Permite seleccionar múltiples intervalos, manteniendo pulsada la tecla CTRL mientras se seleccionan con el ratón uno a uno, o la tecla de mayúsculas, mientras se pulsa el primer elemento y el último de un intervalo.
- ✓ `SINGLE_INTERVAL_SELECTION`: Permite seleccionar un único intervalo, manteniendo pulsada la tecla mayúsculas mientras se selecciona el primer y último elemento del intervalo.
- ✓ `SINGLE_SELECTION`: Permite seleccionar cada vez un único elemento de la lista.

Podemos establecer el valor de la propiedad `selectedIndex` con el método `setSelectedIndex()` es decir, el índice del elemento seleccionado, para seleccionar el elemento del índice que se le pasa como argumento.

Como hemos comentado, los datos se almacenan en un modelo que al fin y al cabo es un vector, por lo que tiene sentido hablar de índice seleccionado.

También se dispone de un método `getSelectedIndex()` con el que podemos averiguar el índice del elemento seleccionado.

El método `getSelectedValue()` devuelve el objeto seleccionado, de tipo `Object`, sobre el que tendremos que aplicar un “casting” explícito para obtener el elemento que realmente contiene la lista (por ejemplo un `String`).

Observa que la potencia de usar como modelo un vector de `Object`, es que en el `JList` podemos mostrar realmente cualquier cosa, como por ejemplo, una imagen.

El método `setSelectedValue()` permite establecer cuál es el elemento que va a estar seleccionado.

Si se permite hacer selecciones múltiples, contamos con los métodos:

- ✓ `setSelectedIndices()`, al que se le pasa como argumento un vector de enteros que representa los índices a seleccionar.
- ✓ `getSelectedIndices()`, que devuelve un vector de enteros que representa los índices de los elementos o ítems que en ese momento están seleccionados en el `JList`.
- ✓ `getSelectedValues()`, que devuelve un vector de `Object` con los elementos seleccionados en ese momento en el `JList`.

Ejemplo “Copiar una lista a otra”

```
// Cómo copiar elementos de una lista a otra.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PruebaSeleccionMultiple extends JFrame
{
    private JList listaColores, listaCopia;
    private JButton botonCopiar;
    private final String nombresColores[] = { "Negro", "Azul", "Cyan",
        "Gris oscuro", "Gris", "Verde", "Gris claro", "Magenta", "Naranja",
        "Rosa", "Rojo", "Blanco", "Amarillo" };

    // configurar GUI
    public PruebaSeleccionMultiple()
    {
        super( "Listas de selección múltiple" );

        // obtener panel de contenido y establecer su esquema
        Container contenedor = getContentPane();
        contenedor.setLayout( new FlowLayout() );

        // establecer objeto JList listaColores
        listaColores = new JList( nombresColores );
```



```

listaColores.setVisibleRowCount( 5 );
listaColores.setSelectionMode(
    ListSelectionModel.MULTIPLE_INTERVAL_SELECTION );
contenedor.add( new JScrollPane( listaColores ) );

// crear botón copiar y registrar su componente de escucha
botonCopiar = new JButton( "Copiar >>>" );
botonCopiar.addActionListener(

    new ActionListener() { // clase interna anónima

        // manejar evento de botón
        public void actionPerformed((ActionEvent evento) )
        {
            // colocar valores seleccionados en listaCopia
            listaCopia.setListData( listaColores.getSelectedValues() );
        }

    } // fin de clase interna anónima

); // fin de la llamada a addActionListener

contenedor.add( botonCopiar );

// establecer objeto JList listaCopia
listaCopia = new JList( );
listaCopia.setVisibleRowCount( 5 );
listaCopia.setFixedCellWidth( 100 );
listaCopia.setFixedCellHeight( 15 );
listaCopia.setSelectionMode(
    ListSelectionModel.SINGLE_INTERVAL_SELECTION );
contenedor.add( new JScrollPane( listaCopia ) );

setSize( 325, 130 );
setVisible( true );

} // fin del constructor PruebaSeleccionMultiple

public static void main( String args[] )
{
    JFrame.setDefaultLookAndFeelDecorated(true);
    PruebaSeleccionMultiple aplicacion = new PruebaSeleccionMultiple();
    aplicacion.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
}

} // fin de la clase PruebaSeleccionMultiple

```

Los componentes JList permiten la selección de elementos de una lista, siempre que estén uno a continuación del otro de manera secuencial.



Si



No

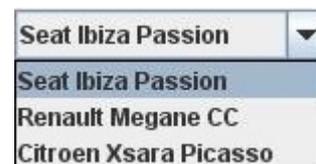
5.9 Listas desplegadas.

Una **lista desplegable** se representa en Java con el componente Swing `JComboBox`. Consiste en una lista en la que sólo se puede elegir una opción.

Se pueden crear JComboBoxes tanto editables como no editables.

Una lista desplegable es una mezcla entre un campo de texto editable y una lista. Si la propiedad **editable** de la lista desplegable la fijamos a verdadero, o sea a **true**, el usuario podrá seleccionar uno de los valores de la lista que se despliega al pulsar el botón de la flecha hacia abajo y dispondrá de la posibilidad de teclear directamente un valor en el campo de texto.

Establecemos la propiedad **editable** del `JComboBox` el método `setEditable()` y se comprueba con el método `isEditable()`. La clase `JComboBox` ofrece una serie de métodos que tienen nombres y funcionalidades similares a los de la clase `JList`.



Puedes ver en el siguiente videotutorial cómo se crea una aplicación con NetBeans, en la que se van añadiendo los controles que hemos visto, entre ellos una lista desplegable.

<http://www.youtube.com/watch?v=EVdjtI8BFu4>

Se ve como con el asistente de NetBeans se crea un nuevo proyecto. En él se crea un formulario y sobre el formulario se arrastra un panel y luego otro, uno debajo del otro, desde la paleta de controles. Entre medias de los dos paneles, se sitúan cuatro botones que se arrastran también desde la paleta.

En el panel de arriba se sitúan dos controles JComboBox arrastrados desde la paleta, y en el panel de abajo se sitúan tres campos de texto. Los botones se cambian de etiquetan para que muestren respectivamente los símbolos de la suma, la resta, la multiplicación y la división.

También, se editan los valores posibles que pueden tomar los combos, para que sean números.

Ejemplo "PruebaJComboBox"

```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.WindowConstants;

/**
 * Ejemplo de uso de JComboBox.
 * @author chuidiang
 */
public class PruebaJComboBox {

    private JTextField tf;
    private JComboBox combo;
    private JFrame v;

    /**
     * @param args
     */
    public static void main(String[] args) {
        new PruebaJComboBox();
    }

    public PruebaJComboBox()
    {
        // Creacion del JTextField
        tf = new JTextField(20);

        // Creacion del JComboBox y añadir los items.
        combo = new JComboBox();
        combo.addItem("uno");
        combo.addItem("dos");
        combo.addItem("tres");

        // Accion a realizar cuando el JComboBox cambia de item seleccionado.
        combo.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                tf.setText(combo.getSelectedItem().toString());
            }
        });

        // Creacion de la ventana con los componentes
        v = new JFrame();
        v.getContentPane().setLayout(new FlowLayout());
        v.getContentPane().add(combo);
        v.getContentPane().add(tf);
        v.pack();
    }
}
```

```

        v.setVisible(true);
        v.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }
}

```

5.10 Menús.

En las aplicaciones informáticas siempre se intenta que el usuario disponga de un menú para facilitar la localización una operación. La filosofía, al crear un menú, es que contenga todas las acciones que el usuario pueda realizar en la aplicación. Lo más normal y útil es hacerlo clasificando o agrupando las operaciones por categorías en submenús.

En Java usamos los componentes `JMenu` y `JMenuItem` para crear un menú e insertarlo en una barra de menús.

La barra de menús es un componente `JMenuBar`. Los constructores son los siguientes:

- ✓ `JMenu()`: Construye un menú sin título.
- ✓ `JMenu(String s)`: Construye un menú con título indicado por s.
- ✓ `JMenuItem()`: Construye una opción sin icono y sin texto.
- ✓ `JMenuItem(Icon icono)`: Construye una opción con icono y con texto.

Podemos construir por código un menú sencillo como el de la imagen con las siguientes sentencias:

```

// Crear la barra de menú
JMenuBar barra = new JMenuBar();
// Crear el menú Archivo
JMenu menu = new JMenu("Archivo");
// Crear las opciones del menú
JMenuItem opcionAbrir = new JMenuItem("Abrir");
menu.add(opcionAbrir);
JMenuItem opcionguardar = new JMenuItem("Guardar");
menu.add(opcionguardar);
JMenuItem opcionSalir = new JMenuItem("Salir");
menu.add(opcionSalir);
// Añadir las opciones a la barra
barra.add(menu);
// Establecer la barra
setJMenuBar(barra);

```

Frecuentemente, dispondremos de alguna opción dentro de un menú, que al elegirla, nos dará paso a un conjunto más amplio de opciones posibles.

Cuando en un menú, un elemento del mismo es a su vez un menú, se indica con el símbolo al final de esa opción, de forma que se sepa que, al seleccionarla, nos abrirá un nuevo menú.

Para incluir un menú como submenú de otro basta con incluir como ítem del menú, un objeto que también sea un menú, es decir, incluir dentro del `JMenu` un elemento de tipo `JMenu`.

Ejemplo "PruebaMenu"

```

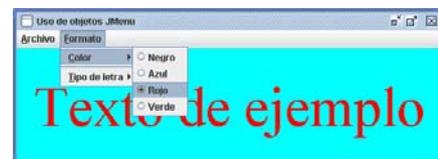
// Demostración de los menús
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PruebaMenu extends JFrame
{
    private final Color valoresColor[]={Color.black,Color.blue,Color.red,Color.green };
    private JRadioButtonMenuItem elementosColor[], tiposLetra[];
    private JCheckBoxMenuItem elementosEstilo[];
    private JLabel pantallaEtiqueta;
    private ButtonGroup grupoTiposLetra, grupoColores;
    private int estilo;

    // configurar GUI
    public PruebaMenu()
    {
        super( "Uso de objetos JMenu" );

        // establecer menú Archivo y sus elementos de menú
        JMenu menuArchivo = new JMenu( "Archivo" );
        menuArchivo.setMnemonic( 'A' );

```



```

// establecer elemento de menú Acerca de...
JMenuItem elementoAcerca = new JMenuItem( "Acerca de..." );
elementoAcerca.setMnemonic( 'c' );
menuArchivo.add( elementoAcerca );
elementoAcerca.addActionListener(

    new ActionListener() { // clase interna anónima

        // mostrar cuadro de diálogo de mensaje cuando el usuario seleccione Acerca de...
        public void actionPerformed( ActionEvent evento )
        {
            JOptionPane.showMessageDialog( PruebaMenu.this,
                "Este es un ejemplo\ndel uso de menús",
                "Acerca de", JOptionPane.PLAIN_MESSAGE );
        }

    } // fin de la clase interna anónima

); // fin de la llamada a addActionListener

// establecer elemento de menú Salir
JMenuItem elementoSalir = new JMenuItem( "Salir" );
elementoSalir.setMnemonic( 'S' );
menuArchivo.add( elementoSalir );
elementoSalir.addActionListener(

    new ActionListener() { // clase interna anónima

        // terminar la aplicación cuando el usuario haga clic en elementoSalir
        public void actionPerformed( ActionEvent evento )
        {
            System.exit( 0 );
        }

    } // fin de la clase interna anónima

); // fin de la llamada a addActionListener

// crear barra de menús y adjuntarla a la ventana PruebaMenu
JMenuBar barra = new JMenuBar();
setJMenuBar( barra );
barra.add( menuArchivo );

// crear menú Formato, con sus submenús y elementos de menú
JMenu menuFormato = new JMenu( "Formato" );
menuFormato.setMnemonic( 'F' );

// crear submenú Color
String colores[] = { "Negro", "Azul", "Rojo", "Verde" };

JMenu menuColor = new JMenu( "Color" );
menuColor.setMnemonic( 'C' );

elementosColor = new JRadioButtonMenuItem[ colores.length ];
grupoColores = new ButtonGroup();
ManejadorEventos manejadorEventos = new ManejadorEventos();

// crear elementos de menú tipo botones de opción para el menú Color
for ( int cuenta = 0; cuenta < colores.length; cuenta++ ) {
    elementosColor[ cuenta ] =
        new JRadioButtonMenuItem( colores[ cuenta ] );
    menuColor.add( elementosColor[ cuenta ] );
    grupoColores.add( elementosColor[ cuenta ] );
    elementosColor[ cuenta ].addActionListener( manejadorEventos );
}

// seleccionar primer elemento del menú Color
elementosColor[ 0 ].setSelected( true );

// agregar el menú Formato a la barra de menús
menuFormato.add( menuColor );
menuFormato.addSeparator();

// crear submenú Tipo de letra
String nombresTiposLetra[] = { "Serif", "Monospaced", "SansSerif" };

JMenu menuTiposLetra = new JMenu( "Tipo de letra" );

```

```

menuTiposLetra.setMnemonic( 'T' );

tiposLetra = new JRadioButtonMenuItem[ nombresTiposLetra.length ];
grupoTiposLetra = new ButtonGroup();

// crear elementos de menú tipo botones de opción para el menú Tipos de letra
for ( int cuenta = 0; cuenta < tiposLetra.length; cuenta++ ) {
    tiposLetra[ cuenta ] = new JRadioButtonMenuItem( nombresTiposLetra[ cuenta ] );
    menuTiposLetra.add( tiposLetra[ cuenta ] );
    grupoTiposLetra.add( tiposLetra[ cuenta ] );
    tiposLetra[ cuenta ].addActionListener( manejadorEventos );
}

// seleccionar el primer elemento del menú Tipo de letra
tiposLetra[ 0 ].setSelected( true );

menuTiposLetra.addSeparator();

// establecer elementos del menú Estilo
String nombresEstilo[] = { "Negrita", "Cursiva" };

elementosEstilo = new JCheckBoxMenuItem[ nombresEstilo.length ];
ManejadorEstilo manejadorEstilo = new ManejadorEstilo();

// crear elementos de menú tipo casilla de verificación para el menú Estilo
for ( int cuenta = 0; cuenta < nombresEstilo.length; cuenta++ ) {
    elementosEstilo[ cuenta ] =
        new JCheckBoxMenuItem( nombresEstilo[ cuenta ] );
    menuTiposLetra.add( elementosEstilo[ cuenta ] );
    elementosEstilo[ cuenta ].addItemListener( manejadorEstilo );
}

// colocar menú Tipo de letra en el menú Formato
menuFormato.add( menuTiposLetra );

// agregar menú Formato a la barra de menús
barra.add( menuFormato );

// establecer etiqueta para mostrar texto
pantallaEtiqueta = new JLabel( "Texto de ejemplo", SwingConstants.CENTER );
pantallaEtiqueta.setForeground( valoresColor[ 0 ] );
pantallaEtiqueta.setFont( new Font( "Serif", Font.PLAIN, 72 ) );

getContentPane().setBackground( Color.CYAN );
getContentPane().add( pantallaEtiqueta, BorderLayout.CENTER );

setSize( 550, 200 );
setVisible( true );
} // fin del constructor

public static void main( String args[] )
{
    JFrame.setDefaultLookAndFeelDecorated(true);
    PruebaMenu aplicacion = new PruebaMenu();
    aplicacion.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
}

// clase interna para manejar eventos de acción de los elementos de menú
private class ManejadorEventos implements ActionListener {

    // procesar selecciones de color y tipo de letra
    public void actionPerformed( ActionEvent evento )
    {
        // procesar selección de color
        for ( int cuenta = 0; cuenta < elementosColor.length; cuenta++ )

            if ( elementosColor[ cuenta ].isSelected() ) {
                pantallaEtiqueta.setForeground( valoresColor[ cuenta ] );
                break;
            }

        // procesar selección de tipo de letra
        for ( int cuenta = 0; cuenta < tiposLetra.length; cuenta++ )

            if ( evento.getSource() == tiposLetra[ cuenta ] ) {
                pantallaEtiqueta.setFont(

```

```

        new Font( tiposLetra[ cuenta ].getText(), estilo, 72 ) );
        break;
    }

    repaint();

} // fin del método actionPerformed

} // fin de la clase ManejadorEventos

// clase interna para manejar eventos de los elementos de menú tipo casilla de verificación
private class ManejadorEstilo implements ItemListener {

    // procesar selecciones de estilo de tipo de letra
    public void itemStateChanged( ItemEvent e )
    {
        estilo = 0;

        // checar selección de negrita
        if ( elementosEstilo[ 0 ].isSelected() )
            estilo += Font.BOLD;

        // checar selección de cursiva
        if ( elementosEstilo[ 1 ].isSelected() )
            estilo += Font.ITALIC;

        pantallaEtiqueta.setFont(
            new Font( pantallaEtiqueta.getFont().getName(), estilo, 72 ) );

        repaint();
    }

} // fin de la clase ManejadorEstilo

} // fin de la clase PruebaMenu

```

Un menú se crea utilizando el componente JMenu dentro de un JList



5.10.1 Separadores.

A veces, en un menú pueden aparecer distintas opciones. Por ello, nos puede interesar destacar un determinado grupo de opciones o elementos del menú como relacionados entre sí por referirse a un mismo tema, o simplemente para separarlos de otros que no tienen ninguna relación con ellos.

El componente Swing que tenemos en Java para esta funcionalidad es:

`JSeparator`, que dibuja una línea horizontal en el menú, y así separa visualmente en dos partes a los componentes de ese menú. En la imagen podemos ver cómo se han separado las opciones de Abrir y Guardar de la opción de Salir, mediante este componente.

Al código anterior, tan sólo habría que añadirle una línea, la que resaltamos en negrita:

```

...
menu.add(opcionguardar);
menu.add(new JSeparator());
JMenuItem opcionSalir = new JMenuItem("Salir");
...

```



En un menú en Java se debe introducir siempre un separador para que se pueda compilar el código



En este enlace podrás encontrar más información sobre diseño de menús.

http://biblioteca.uns.edu.pe/saladocentes/archivoz/publicacionez/Programacion_Visual_con_Java_2.pdf

5.10.2 Aceleradores de teclado y mnemónicos.

A veces, tenemos que usar una aplicación con interfaz gráfica y no disponemos de ratón, porque se nos ha roto, o cualquier causa.

Además, cuando diseñamos una aplicación, debemos preocuparnos de las características de accesibilidad.

Algunas empresas de desarrollo de software obligan a todos sus empleados a trabajar sin ratón al menos un día al año, para obligarles a tomar conciencia de la importancia de que todas sus aplicaciones deben ser accesibles, usables sin disponer de ratón.

Ya no sólo en menús, sino en cualquier componente interactivo de una aplicación: campo de texto, botón de acción, lista desplegable, etc, es muy recomendable que pueda seleccionarse sin el ratón, haciendo uso exclusivamente del teclado.

Para conseguir que nuestros menús sean accesibles mediante el teclado, la idea es usar aceleradores de teclado o **atajos de teclado** y de **mnemónicos**.

Un acelerador o **atajo de teclado** es una combinación de teclas que se asocia a una opción del menú, de forma que pulsándola se consigue el mismo efecto que abriendo el menú y seleccionando esa opción.

Esa combinación de teclas **aparece escrita a la derecha de la opción del menú**, de forma que el usuario pueda tener conocimiento de su existencia.

Para añadir un atajo de teclado, lo que se hace es emplear la propiedad `accelerator` en el diseñador. Los atajos de teclado **no suelen** asignarse a todas las opciones del menú, sino **sólo a** las que más se usan.

Un **mnemónico** consiste en resaltar una tecla dentro de esa opción, mediante un subrayado, de forma que pulsando `Alt + <el mnemónico>` se abra el menú correspondiente. Por ejemplo en la imagen con `Alt + A` abriríamos ese menú que se ve, y ahora con `Ctrl+G` se guardaría el documento.

Para añadir mediante código un mnemónico a una opción del menú, se hace mediante la **propiedad** `mnemonic`.

Los mnemónicos sí que deberían ser incluidos para todas las opciones del menú, de forma que todas puedan ser elegidas haciendo uso sólo del teclado, mejorando así la accesibilidad de nuestra aplicación. Para ver cómo se añadiría mediante el diseñador de NetBeans, mira el siguiente. Debes conocer.



En el siguiente enlace se puede ver cómo añadir a una aplicación que estemos construyendo con NetBeans, entre otras cosas, mnemónicos y aceleradores

<http://itagua.wordpress.com/2010/09/24/tutorial-de-java-swing-11-imenubar-imenubarmenuitemprogressbar-jtextarea/>